Math

UIUCDCS-R-75-716

A STUDY OF CONCRETE COMPUTATIONAL COMPLEXITY

by

Andrew Chi-Chih Yao

May 1975

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-75-716

A STUDY OF CONCRETE COMPUTATIONAL COMPLEXITY*

by

Andrew Chi-Chih Yao

May 1975

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

*This work was supported in part by the Department of Computer Science and in part by the National Science Foundation under Grant GJ-41538, and was submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science, 1975.

ACKNOWLEDGMENT

I am very grateful to my thesis advisor, Professor Chung-Laung Liu, for the invaluable guidance, advice and encouragement he gave during my study at the University of Illinois. I would also like to thank Professor Edward Reingold for the genuine interest and kind aid extended to me.

Thanks are due to the Computer Science Department of the University of Illinois and the National Science Foundation for supporting this research. Special thanks go to Mrs. Connie Slovak for the excellent work in typing this thesis as well as the earlier manuscripts.

I wish to thank my wife, Frances, who introduced me to this fascinating field of study, provided many ideas and suggestions related to this work, and constantly devoted her love. Finally, I am happy to have had a most pleasant time with my officemates, Jim Bitner, Nai-Fung Chen, Brian Hansche, and John Koch.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 Concrete Computational Complexity

A great variety of computational problems are encountered in computer science. Some arise in the design of computer systems, e.g., job scheduling, connection networks; others come from different disciplines, but are to be solved on a computer, e.g., solutions of systems of equations, graph problems. In all cases, good algorithms are needed in order to efficiently use the available resources. The study of the complexity of these computations constitutes the realm of concrete computational complexity.

In complexity theory the goal is to find good algorithms and prove that they are close to the best possible. Thus, there are two complementary facets to the complexity problem. One aspect is to find better and better algorithms and analyze their costs, which gives us upper bounds to the "difficulty" of the problem considered. The other aspect is to establish lower bounds on the minimum cost necessary for all algorithms. When the upper and lower bounds come close together, we would then have a clear measure of the intrinsic complexity of a problem.

1.2 Models of Computation

For any given problem, a computational model has to be set up so that the complexity of the problem can be discussed in a precise way. This means specifying a repertoire of elementary operations, which

defines the class of algorithms allowed; and a complexity measure, which quantifies the concept of cost of algorithms. The complexity of the problem, the main object of interest, is then defined as the minimum cost required by all algorithms. To make the results meaningful, the choice of the class of algorithms should be wide enough, and the cost measure should reflect the "true cost", e.g., running time of programs, in the real world.

A prominent example is the decision tree model for the problem of sorting a set of numbers $\{x_1, x_2, \ldots, x_n\}$. In this model, an algorithm consists of a finite sequence of queries $x_{i_1} > x_{i_2}$?, $x_{i_3} > x_{i_4}$?, ..., where the choice of a query may depend on the answers to the previous queries. The cost of an algorithm is defined as the maximum number of queries asked over all possible inputs. It is well known [17] that there are algorithms with cost $\approx n\log_2 n$, and that any algorithm must have a cost greater than or equal to $\log_2(n!) \approx n\log_2 n$. Therefore, the complexity $S(n)$ of this problem is approximately $n\log_2 n$.

It should be emphasized that care must be taken when practical conclusions are drawn from results in complexity theory. There is always the pitfall of having an over-simplified model. Furthermore, many theoretical results are statements about asymptotic behavior while the problem at hand may not have the size for the theorems to be interesting.

1.3 An Overview of the Thesis

We shall study the complexity of four problems, one belonging to each of the following categories:

(i) Discrete Problem: Selection networks

(ii) Continuous Problem: Computing the minima of quadratic forms

(iii) Graph Problem: Finding minimum spanning trees

(iv) Heuristic Algorithm: Scheduling unit time tasks with limited resources

A summary of the results follows. Preliminary versions for these results have appeared in [27], [28], [29], and [30].

Selection Networks (Chapter 2) [27]

We investigate the complexity of network selection by measuring it in terms of $U(t,N)$, the minimum number of comparators needed, and $T(t,N)$, the minimum delay time possible, for networks selecting the smallest t elements from a set of N inputs. New bounds on $U(t,N)$ and $T(t,N)$ are obtained. In particular, $U(3,N)$ is determined to within a constant of 2, and asymptotic formulae for $U(t,N)$ and $T(t,N)$ are obtained for fixed t.

Computing the Minima of Quadratic Forms (Chapter 3) [28]

Let $F(x_1,x_2,\ldots,x_n)$ be a quadratic form in n variables. We wish to compute the point $x^{(0)}$ at which F achieves its minimum, by a series of adaptive functional evaluations. We prove that $O(n^2)$ evaluations are necessary in the worst case for any such algorithm.

Finding Minimum Spanning Trees (Chapter 4) [29]

We obtained an algorithm which finds a minimum spanning tree in $O(|E|\log\log|V|)$ time for a graph $G = (V,E)$. Previously the best algorithms known have running time $O(|E|\sqrt{\log|V|})$ for sparse graphs.

Scheduling Unit-Time Tasks with Limited Resources (Chapter 5) [30]

A set of tasks are to be scheduled on a multiprocessing system with s resources. Each task takes one unit time to complete, and rquires certain amounts of resources. The schedule is to be consistent with a prescribed partial order relation on the tasks, and the total demand for each resource must not exceed a fixed amount at any instant. We analyze the worst-case behavior of several heuristic scheduling algorithms.

Let ω be the time taken for executing all the tasks according to a priority list, and ω_0 be the time required when scheduled in an optimal way. It is shown that, independent of the number of processors, $\omega/\omega_0 \leqq s\omega_0/2 + O(s)$ for any list. When certain heuristic algorithms are used to prepare the list, a significantly improved upper bound can be derived: $\omega/\omega_0 \leqq \text{const.} \times s + O(1)$. Some generalizations are possible to the case when the "unit-time" restriction is removed.

When the partial order relation is empty, the problem becomes a natural generalization of the bin-packing problem. A bound of $\omega/\omega_0 \leqq s + \frac{17}{20} + \epsilon$ is given.

2. SELECTION NETWORKS

2.1 Introduction

A (t,N)-selector where $1 \leq t \leq N$ is a network with N inputs $(x_1, x_2, \ldots, x_N)$ and N outputs $(x_1', x_2', \ldots, x_N')$ such that the set $\{x_1', x_2', \ldots, x_t'\}$ consists of the smallest t elements of $x_1, x_2, \ldots, x_N$. We consider (t,N)-selectors that are built of basic modules called comparators which are themselves (1,2)-selectors. A (2,5)-selector is shown in Figure 2.1 where a comparator is represented as

with $y_1' = \min(y_1, y_2)$ and $y_2' = \max(y_1, y_2)$. Note that the "sorting networks" (for N elements), which have been extensively studied [7], [14], [18], [24], are networks that are (t,N)-selectors for all t $(1 \leq t \leq N)$.

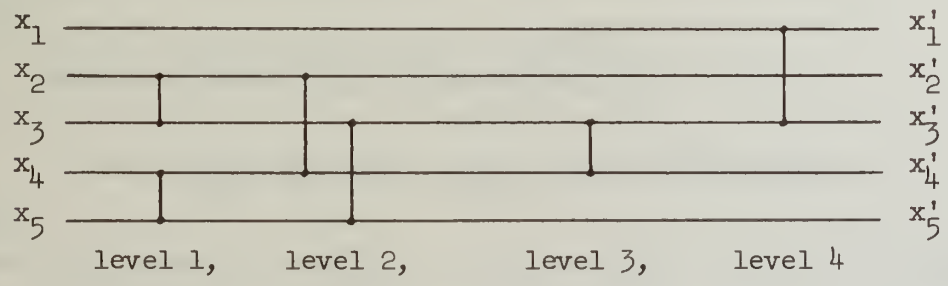


Figure 2.1 A (2,5)-selector.

In a network the comparators may be grouped into levels, where within each level are some comparisons that can be made simultaneously (see Figure 2.1). The delay time of a network is the minimum number of levels that its comparators can be grouped into.

In this chapter, we investigate the complexity of network selection by measuring it in terms of $U(t,N)$, the minimum number of comparators needed in a (t,N)-selector, and $T(t,N)$, the minimum delay time possible for (t,N)-selectors. New bounds on $U(t,N)$ and $T(t,N)$ are presented. In particular, $U(3,N)$ is determined to within a constant of 2, and asymptotic formulae for $U(t,N)$ and $T(t,N)$ are given for fixed t. A new lower bound on the delay time for sorting networks is also obtained. Main results are contained in Theorems 2.1, 2.3, 2.7, 2.8, 2.11.

2.2 Known Bounds for $U(t,N)$ and $T(t,N)$

In the literature, the following results (A), (B), (C) are known:

(A) [18] $U(1,N) = N-1, \quad U(2,N) = 2N-4$ (2.1)

(B) (Alexeyev, see reference 1)

$$(N-t)\lceil \log(t+1) \rceil \leq U(t,N) \leq (N-t)\left(1 + \frac{2\hat{S}(t)}{t}\right) \tag{2.2}$$

where $\hat{S}(t)$ is the minimum number of comparators needed in a sorting network with t inputs.

(C) [26] $$T(t,N) \geq \frac{1}{\log 3 - 1}\left(\log t + \log\left(1 - \frac{t}{N}\right)\right) \tag{2.3}$$

Remark Throughout this chapter, logarithms are taken with respect to base 2.

2.3 New Results Concerning U(t,N)

In this section, we shall show that U(t,N) is well approximated by $\lceil \log(t+1) \rceil N$ when t is small compared to $\sqrt{N}$. A new lower bound for U(t,N), which improves on Alexeyev's bound (2.2) in most cases, is also derived. As a consequence, U(3,N) is determined to within a constant.

2.3.1 Asymptotic Behavior of U(t,N) for Fixed t

Since the best upper bound known for $\hat{S}(t)$ is of the order $t(\log t)^2$ for general t, the inequality that can be deduced from (2.2) is

$$\lceil \log(t+1) \rceil (N-t) \leq U(t,N) \leq C(\log t)^2 N \tag{2.4}$$

where C is a constant. Thus, the asymptotic behavior of U(t,N) for fixed t is not well determined by (2.2). We shall construct (t,N)-selectors that yield a better upper bound for U(t,N). This enables us to identify the leading term of U(t,N). For example, when t = 11, we can obtain

$$4(N-11) \leq U(11,N) \leq 4N + C((\log N)^2) \tag{2.5}$$

for some constant C. In general, we have

Theorem 2.1

$$U(t,N) = \lceil \log(t+1) \rceil N + O((\log N)^{\lceil \log \frac{t+1}{3} \rceil}) \text{ for fixed } t. \tag{2.6}$$

To prove Theorem 2.1 we need the following lemma.

Lemma 2.2

$$U(t,N) \leq U(\lfloor t/2 \rfloor, \lfloor N/2 \rfloor) + U(t, \lceil \tfrac{N}{2} \rceil + \lfloor t/2 \rfloor) + \lfloor N/2 \rfloor . \tag{2.7}$$

Proof of Lemma 2.2 We need only show that a (t,N)-selector can be constructed from one $(\lfloor t/2 \rfloor, \lfloor N/2 \rfloor)$-selector, one $(t, \lceil N/2 \rceil + \lfloor t/2 \rfloor)$-selector,

and $\lfloor N/2 \rfloor$ additional comparators. Figure 2.2 shows such a construction. The reason that it works as a (t,N)-selector is that, after the initial $\lfloor N/2 \rfloor$ comparisons in A, at most $\lfloor t/2 \rfloor$ of the smallest t elements can come out on the $\lfloor N/2 \rfloor$ lines of the lower half. These possible "small" elements are selected by B and input into C. Therefore all of the smallest t elements are fed into the $(t, \lceil N/2 \rceil + \lfloor t/2 \rfloor)$-selector C which finally outputs those elements on the top t output lines. This proves Lemma 2.2. □

Proof of Theorem 2.1 According to Equation (2.1),

$$U(1,N) = N-1, \quad U(2,N) = 2N-4. \tag{2.8}$$

Using these equations as the basis of induction, it is not hard to prove (2.6) from (2.7). □

It is clear that (t,N)-selectors which satisfy the bound of Theorem 2.1 can be explicitly constructed by following inductive scheme illustrated in Figure 2.2, based on $(1,N)$-selectors and $(2,N)$-selectors which achieve (2.8). As will be seen in Section 2.3.3, the $(3,N)$-selectors thus constructed are optimal within a constant number of comparators.

2.3.2 Other Sufficient Conditions for $U(t,N) \approx \lceil \log(t+1) \rceil N$

According to Theorem 2.1, $\lceil \log(t+1) \rceil N$ is the dominant term of $U(t,N)$ for fixed t as $N \to \infty$. Actually, this is true under more general situations. For example, we have the following theorem:

Theorem 2.3 If $f(N)$ satisfies $\lim_{N\to\infty} \frac{f(N)}{N^{\frac{1}{2}-\epsilon}} = 0$ for some $\epsilon > 0$, then

$$\lim_{N\to\infty} \frac{U(f(N),N)}{N\lceil \log(f(N)+1) \rceil} = 1 \tag{2.9}$$

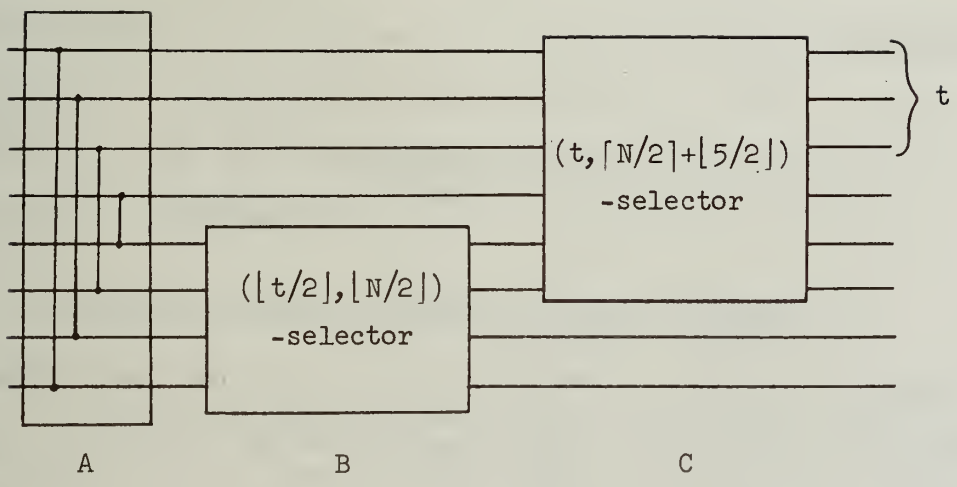


Figure 2.2 A (5,N)-selector.

Corollary 2.3 If $0 < \alpha < 1/2$, then

$$\lim_{N\to\infty} \frac{U(N^{\alpha}, N)}{N \log N} = \alpha \tag{2.10}$$

Thus, for any t satisfying $t = o(N^{\frac{1}{2}-\epsilon})$, $U(t,N)$ is well approximated by $\lceil \log(t+1) \rceil N$. The validity of (2.9) and (2.10) are direct consequences of the following theorem and Alexeyev's lower bound $\lceil \log(t+1) \rceil (N-t)$ for $U(t,N)$.

Theorem 2.4 For $t < \sqrt{N}$,

$$U(t,N) \leq \lceil \log(t+1) \rceil N + 4 \lceil \log(\frac{t+1}{3}) \rceil^3 \begin{pmatrix} \lceil \log N \rceil \\ \lceil \log \frac{t+1}{3} \rceil \end{pmatrix} \tag{2.11}$$

The proof of Theorem 2.4 will be given in Appendix A. Obviously, Theorem 2.1 is implied by (2.11). We have proved Theorem 2.1 separately because it has a more elegant proof of its own, also the construction described there often yields better networks than obtained by the general construction of Theorem 2.4.

2.3.3 Lower Bounds for $U(t,N)$

Since $U(1,N) = N-1$ and $U(2,N) = 2(N-2)$, it is an interesting question whether Alexeyev's lower bound $\lceil \log(t+1) \rceil (N-t)$ is in general achievable for $U(t,N)$. We shall give a new lower bound which shows that, for most values of t, Alexeyev's lower bound cannot be achieved.

The new lower bound that improves on Alexeyev's lower bound (2.2) asymptotically whenever $t \neq 2^k$ is given below:

Theorem 2.5 If t is not a power of 2, we have

$$U(t,N) \geq \lceil \log(t+1) \rceil N + (t - 2^{\lfloor \log t \rfloor}) \log N + C(t) \tag{2.12}$$

for some function $C(t)$.

We shall prove Theorem 2.5 by showing it for the special case $t = 3$. The arguments are immediately generalizable to any $t \neq 2^k$ by using induction. Thus we are actually going to show

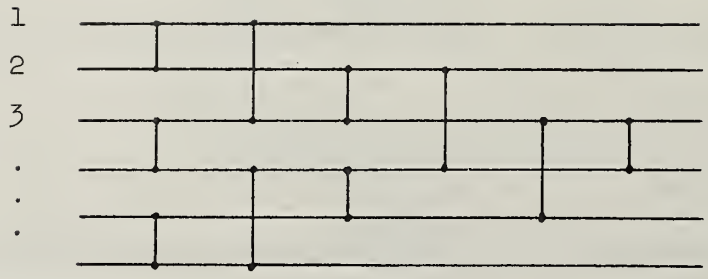
Theorem 2.6

$$U(3,N) \geq 2N-6 + \lceil \log_2 \lceil N/3 \rceil \rceil \tag{2.13}$$

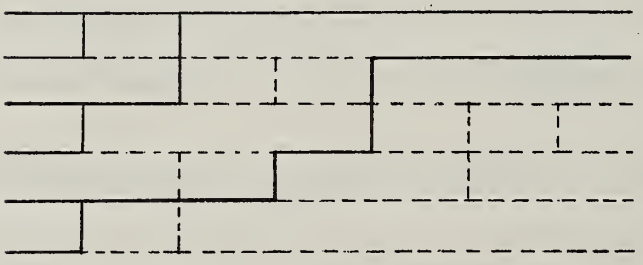
Proof The basic idea of "pruning" a network used in this proof is similar to that used by Van Voohris [25] in showing lower bounds for sorting networks. Let A be any $(3,N)$-selector such as the one shown in Figure 2.3a. We will show that, by removing at least $\lceil \log_2 \lceil N/3 \rceil \rceil$ comparators from A and reconnecting some of the lines, we shall be left with a $(2,N-1)$-selector. This leads to inequality (2.13) since

$$U(3,N) \geq U(2,N-1) + \lceil \log_2 \lceil N/3 \rceil \rceil = 2N-6 + \lceil \log_2 \lceil N/3 \rceil \rceil .$$

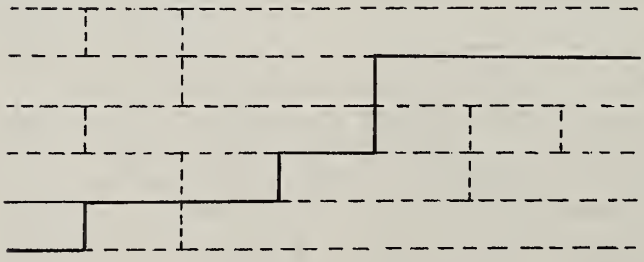
We begin by numbering the lines of the network from the top as in Figure 2.3a. If the smallest element is input to the j-th line, for any $1 \leq j \leq N$, this element will always move "upward" across any comparators encountered, and wind up on one of the top three lines at the output end (see Figure 2.3b). As j runs from 1 to N, these N paths can be divided into three groups according to the output line they lead to. One group will contain at least $\lceil N/3 \rceil$ paths. We can look at this group of paths as a binary tree, regarding the comparators contained in the paths as branch nodes (internal nodes), and the input terminals as leaves. Since there are at least $\lceil N/3 \rceil$ leaves, there is a path with at least $\lceil \log_2 \lceil N/3 \rceil \rceil$ comparators connected to it (Figure 2.3c). We can remove this path and all the comparators incident with it, then reconnect the lines (and straighten them if necessary) as shown in Figure 2.3d. The



(a)

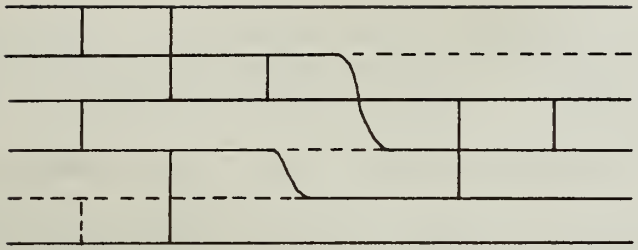


(b)

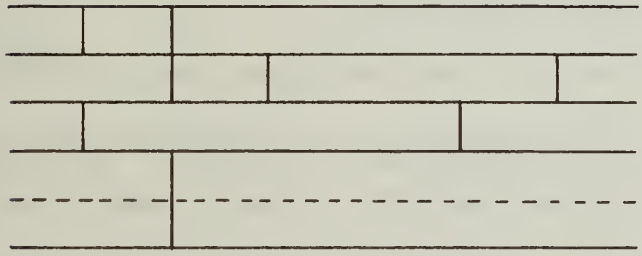


(c)

Figure 2.3 "Pruning" a (3,N)-selector.



(d)



(e)

Figure 2.3 (continued)

new netowrk of Figure 2.3e is the resulting (2,N-1)-selecotr and this proves our claim. □

We conclude this section with the following theorem.

Theorem 2.7

$$2N-6+\lceil \log\lceil N/3\rceil\rceil \leq U(3,N) \leq 2N-5+\lfloor \log(N-3)\rfloor \tag{2.14}$$

Proof We need only prove the upper bound. From Lemma 2.2, we have

$$U(3,N) \leq U(3,\lceil N/2\rceil+1)+U(1,\lfloor N/2\rfloor)+\lfloor N/2\rfloor = U(3,\lceil N/2\rceil+1)+2\lfloor N/2\rfloor-1 \tag{2.15}$$

Using $U(3,5) = 6$ and $U(3,6) = 8$ as the basis of induction, we obtain the inequality on the right-hand side of (2.14). □

2.4 New Results Concerning T(t,N)

In this section a new inequality involving T(t,N) is derived. With the help of this inequality, we can determine the asymptotic value of T(t,N) for fixed t and large N to within a term of the order $\log\log\log N$. For general values of t and N, this inequality also provides lower bounds on T(t,N) that are stronger than those previously known. As an interesting corollary, it is shown that the minimal delay time for a sorting network with N inputs is at least $2.4\log N$ for large N.

2.4.1 Main Theorem

This subsection is devoted to a proof of the following theorem which forms the basis of all later discussions. Throughout this section we adopt the convention that the binomial coefficient $\binom{k}{j}$ is zero if $k < j$.

Theorem 2.8 T(t,N) satisfies the following inequality:

$$t\lceil \log(t+1)\rceil \geq \frac{N}{2^{T(t,N)}} \sum_{i=0}^{\lfloor \log t\rfloor} \left[(\lceil \log(t+1)\rceil - i)\binom{T(t,N)}{i}\right] \tag{2.16}$$

Corollary 2.9

$$T(t,N) \geq \log N + \log\left[\frac{1}{t\lceil \log(t+1)\rceil}\left(\begin{matrix} T(t,N) \\ \lfloor \log t \rfloor \end{matrix}\right)\right] \tag{2.17}$$

Proof of Theorem 2.8 In a network that is divided into s levels, each line can be viewed as being partitioned into s+1 segments as shown in Figure 2.4.

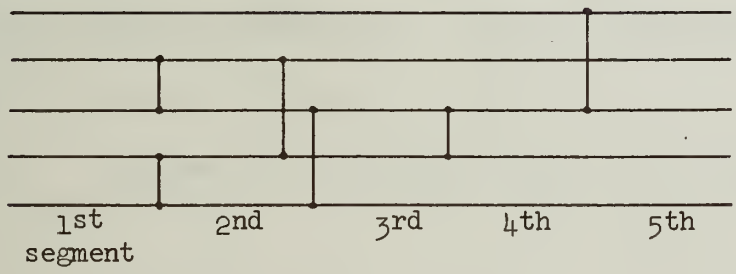


Figure 2.4 Dividing a network into levels.

Let us associate with each line segment a weight as follows [18]:

(1) The first (i.e. the leftmost) segment of each line is assigned weight 0.

(2) If a line is not connected to any comparator at the ℓ^{th} level, then its weights on the ℓ^{th} and the $\ell+1^{st}$ segments are the same.

(3) Let m_i and m_j be the weights on the ℓ^{th} segments of line i and line j respectively where $i < j$ and $\ell \geq 1$. If there is a comparator at the ℓ^{th} level between line i and line j, then on their $\ell+1^{st}$ segments

line i has weight $\min(m_i, m_j)$ and line j has weight $\max(m_i, m_j) + 1$.

An example of the weight assignment of a network is shown in Figure 2.5.

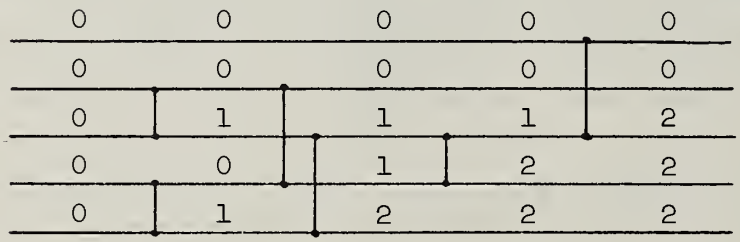


Figure 2.5 Weight assignment of a network.

For a (t,N)-selector with s levels, we define two $s+1$ by $\lfloor \log t \rfloor + 1$ matrices $X = (x_{\ell k})$ and $Y = (y_{\ell k})$ by letting

$x_{\ell k}$ = the number of lines whose ℓ^{th} segments have weight equal to k,

and
$$y_{\ell k} = \sum_{u=0}^{k} (k+1-u)x_{\ell u}, \qquad \ell = 1,2,\ldots,s+1 \qquad (2.18)$$
$$k = 0,1,2,\ldots,\lfloor \log t \rfloor$$

Of course the difference $y_{\ell+1,k} - y_{\ell,k}$ depends very much on the comparators at the ℓ^{th} level. A comparator at the ℓ^{th} level will not affect $y_{\ell+1,k} - y_{\ell,k}$ if at least one of its input segments has weight exceeding k. This is so, because this type of comparator does not change the number of lines with weights u for $u \leq k$. Only those

comparators at the ℓ^{th} level where the maximum weight of two input segments does not exceed k can affect $y_{\ell+1,k}-y_{\ell,k}$. In fact, it is easily seen that every such comparator contributes exactly -1 to the difference $y_{\ell+1,k}-y_{\ell,k}$.

Lemma A $\quad 0 \leq y_{\ell,k}-y_{\ell+1,k} \leq \frac{1}{2}(x_{\ell 0}+x_{\ell 1}+\ldots+x_{\ell k})$

Proof of Lemma A As mentioned above, $y_{\ell,k}-y_{\ell+1,k}$ is equal to the number of comparators at the ℓ^{th} level for which both input line segments have weights not exceeding k. Clearly the number of such comparators is bounded by $\frac{1}{2}(x_{\ell 0}+x_{\ell 1}+\ldots+x_{\ell k})$. □

Lemma B $\quad y_{\ell+1,0} \geq \frac{1}{2} y_{\ell,0}$

$$y_{\ell+1,k} \geq \frac{1}{2}(y_{\ell,k}+y_{\ell,k-1}) \qquad k \geq 1 \tag{2.19}$$

Proof of Lemma B By definition of $y_{\ell k}$,

$$x_{\ell 0} = y_{\ell 0}$$

$$x_{\ell 0}+x_{\ell 1}+\ldots+x_{\ell k} = y_{\ell k}-y_{\ell,k-1} \qquad k \geq 1.$$

According to Lemma A, we have

$$y_{\ell,0}-y_{\ell+1,0} \leq \frac{1}{2} y_{\ell,0}$$

and $$y_{\ell,k}-y_{\ell+1,k} \leq \frac{1}{2}(y_{\ell k}-y_{\ell,k-1}) \qquad k \geq 1.$$

This then leads to (2.19). □

Lemma C $\quad y_{\ell j} \geq \frac{N}{2^{\ell-1}} \sum_{i=0}^{j} \binom{\ell-1}{i} (j+1-i) \qquad \ell = 1,2,\ldots,5+1$

$$j = 0,1,2,\ldots,\lfloor \log t \rfloor$$

Proof of Lemma C Use Lemma B and prove by induction. □

We are now ready to prove Theorem 2.8. As was shown in reference [15], when weight function is so defined, in a (t,N)-selector there are at most t output lines which have weight less than or equal to $\lfloor \log t \rfloor$. Therefore,

$$y_{s+1,\lfloor \log t \rfloor} = \sum_{i=0}^{\lfloor \log t \rfloor} (\lfloor \log t \rfloor + 1 - i) x_{s+1,i}$$

$$\leq (\lfloor \log t \rfloor + t)t = \lceil \log (t+1) \rceil t \tag{2.20}$$

On the other hand, according to Lemma C

$$y_{s+1,\lfloor \log t \rfloor} \geq \frac{N}{2^s} \sum_{i=0}^{\lfloor \log t \rfloor} (\lfloor \log t \rfloor + 1 - i)\binom{s}{i} \tag{2.21}$$

Comparing (2.20) and (2.21), we obtain

$$t\lceil \log (t+1) \rceil \geq \frac{N}{2^s} \sum_{i=0}^{\lfloor \log t \rfloor} (\lceil \log (t+1) \rceil - i)\binom{s}{i} \tag{2.22}$$

This completes the proof of Theorem 2.8. □

2.4.2 Value of $T(t,N)$ for Fixed t and Large N

It is easy to transform a (t,N)-selector into a $(t-t_0, N-t_0)$-selector by deleting the top t_0 lines together with all the comparators that are connected to them. This leads to,

$$T(t,N) \geq T(t-t_0, N-t_0) \tag{2.23}$$

In particular, when $t \leq N/2$, we have

$$T(t,N) \geq T(1,N-t+1) \geq \lceil \log (N-t+1) \rceil \geq \lfloor \log N \rfloor \tag{2.24}$$

To derive a better bound than (2.14) we use equation (2.17),

$$T(t,N) \geq \log N + \log\left[\frac{1}{t\lceil \log(t+1)\rceil}\binom{T(t,N)}{\lfloor \log t \rfloor}\right] \tag{2.25}$$

Now, according to (2.24), $T(t,N) \geq \lfloor \log N \rfloor$. Thus, (2.25) implies

$$T(t,N) \geq \log N + \log\left[\frac{1}{t\lceil \log(t+1)\rceil}\binom{\lfloor \log N \rfloor}{\lfloor \log t \rfloor}\right] \tag{2.26}$$

For fixed t and large N,

$$\binom{\lfloor \log N \rfloor}{\lfloor \log t \rfloor} \sim \frac{(\lfloor \log N \rfloor)^{\lfloor \log t \rfloor}}{(\lfloor \log t \rfloor)!}$$

Therefore, (2.26) becomes

$$T(t,N) \geq \log N + \lfloor \log t \rfloor \log\log N + C(t)$$
$$\text{for some function } C(t) \tag{2.27}$$

The next theorem states that the lower bound in (2.27) is actually a good approximation to $T(t,N)$ in the asymptotic sense.

Theorem 2.10 For any fixed t and large N,

$$T(2,N) = \log N + \log\log N + O(1)$$
$$T(t,N) = \log N + \lfloor \log t \rfloor \log\log N + O(\log\log\log N)$$

Because of (2.27), the theorem can be proved by exhibiting (t,N)-selectors that have the desired number of levels. The construction is similar to that used in the proof of Theorem 2.2. □

2.4.3 More Applications of Theorem 2.8

We shall develop a technique which generates lower bounds for $T(t,N)$ from formula (2.16). Let us state Theorem 2.8 in the following form: For given t,N, let s_0 be the smallest integer s satisfying

$$t\lceil \log_q(t+1) \rceil \geq \frac{N}{2^s} \sum_{i=0}^{\lfloor \log_q t \rfloor} (\lceil \log_q(t+1) \rceil - j)\binom{s}{i} \tag{2.28}$$

then $T(t,N) \geq s_0$.

As can be easily verified, the right-hand side of formula (2.28) is a decreasing function of s for fixed t and N. Therefore, any s that violates (2.28) will satisfy $s_0 \geq s$, and, as a consequence, $T(t,N) \geq s$. This leads to the following procedure:

A technique for generating lower bounds: For any given t,N, find an s violating (2.28). We then have $T(t,N) \geq s$.

To see how this technique works, we prove the following theorem.

Theorem 2.11 For $\frac{N}{2\log_q N} \geq t \geq 1$,

$$T(t,N) \geq 2\lfloor \log_q t \rfloor \tag{2.29}$$

Proof Let $s = 2\lfloor \log_q t \rfloor$.

The right-hand side of (2.28) is equal to

$$\frac{N}{2^s} \sum_{i=0}^{\lfloor \log_q t \rfloor} (\lceil \log_q(t+1) \rceil - i)\binom{s}{i} > \frac{N}{2^s} \sum_{i=0}^{\lfloor \log_q t \rfloor} \binom{s}{i}$$

$$\geq \frac{N}{2^s} \frac{1}{2} \sum_{i=0}^{s} \binom{s}{i} = \frac{N}{2}$$

The left-hand side of (2.28) is equal to

$$t\lceil \log(t+1)\rceil \leq \frac{N}{2\log N}\,(\log N - \log\log N) < \frac{N}{2}$$

Thus (2.28) is not true for $s = 2\lfloor \log t\rfloor$. This proves $T(t,N) \geq 2\lfloor \log t\rfloor$. □

When $t > \frac{N}{2\log N}$, we can first use (2.23) to obtain

$$T(t,N) \geq T\left(\frac{N}{4\log N},\ N-\left(t-\frac{N}{4\log N}\right)\right) \tag{2.30}$$

We now apply Theorem 2.11 to the right-hand side of (2.30) and get

$$T(t,N) \geq 2\left\lfloor \log \frac{N}{4\log N}\right\rfloor$$

Therefore $T(t,N) \geq 2(\log N - \log\log N - 3)$ for $t \geq \frac{N}{2\log N}$. (2.31)

(2.29) and (2.31) constitute a pair of inequalities that are stronger than F. Yao's inequality (2.3).

It should be pointed out that a slightly weaker form of (2.29) can be derived in another way. Noting that each level can contain at most N/2 comparators, we can combine the inequality $T(t,N) \geq \frac{U(t,N)}{(N/2)}$ with Alexeyev's lower bound (2.2) to obtain a result similar to (2.29).

The following theorem is a more interesting application of our technique.

Theorem 2.12 Let $\alpha_0 = \frac{1}{3(2-\log 3)} \approx 0.8$. For any fixed $\epsilon > 0$, there exists a number $f(\epsilon)$ such that

$$T(N^{\alpha_0},N) \geq \left(\frac{1}{2-\log 3} - \epsilon\right)\log N \approx (2.41 - \epsilon)\log N$$

$$\text{for all } N \geq f(\epsilon) \tag{2.32}$$

Corollary 2.13 There exists a function $g(\epsilon)$ such that,

$$T(t,N) \geq \left(\frac{1}{2-\log 3} - \epsilon\right)\log N \text{ for all } N \geq g(\epsilon) \text{ and } N/2 \geq t \geq N^{\alpha_0}. \tag{2.33}$$

Proof of Theorem 2.12 Let $\epsilon_0 > 0$ be any fixed number. Without loss of generality we can assume ϵ satisfies $\epsilon_0 > \epsilon > 0$. We shall choose ϵ_0 to be small enough such that

$$h(\epsilon) > 0 \quad \text{for all } \epsilon_0 > \epsilon > 0 \tag{2.34}$$

where $h(\epsilon) \equiv \epsilon[1-\log(3-\frac{\epsilon}{\alpha_0}) + \log(2-\frac{\epsilon}{\alpha_0})]$

$$+ \alpha_0(3\log(1-\frac{\epsilon}{3\alpha_0}) - 2\log(1-\frac{\epsilon}{2\alpha_0})) \tag{2.35}$$

Now, let $t = N^{\alpha_0}$, and $s = \lceil(\frac{1}{2-\log 3}-\epsilon)\log N\rceil = \lceil 3\alpha_0-\epsilon)\log N\rceil$. We shall prove that (2.28) is not satisfied when N is sufficiently large. This then implies the theorem.

To prove our assertion, we observe that for large N, Stirling's approximation yields

$$\binom{\lceil(3\alpha_0-\epsilon)\log N\rceil}{\lfloor\alpha_0\log N\rfloor} \approx \frac{((3\alpha_0-\epsilon)\log N)!}{(\alpha_0\log N)!\,((2\alpha_0-\epsilon)\log N)!}$$

$$\approx \frac{\sqrt{2\Pi(3\alpha_0-\epsilon)\log N}\left(\frac{(3\alpha_0-\epsilon)\log N}{e}\right)^{(3\alpha_0-\epsilon)\log N}}{\sqrt{2\Pi\alpha_0\log N}\left(\frac{\alpha_0\log N}{e}\right)^{\alpha_0\log N}\sqrt{2\Pi(2\alpha_0-\epsilon)\log N}\left(\frac{(2\alpha_0-\epsilon)\log N}{e}\right)^{(2\alpha_0-\epsilon)\log N}}$$

$$\approx \text{constant} \times \frac{1}{\sqrt{\log N}}\, N^{(3\alpha_0-\epsilon)\log(3\alpha_0-\epsilon)-\alpha_0\log\alpha_0-(2\alpha_0-\epsilon)\log(2\alpha_0-\epsilon)}$$

Therefore, in Formula (2.28)

$$\text{right-hand side} \geq \frac{N}{2^s}\binom{\lceil (3\alpha_0-\epsilon)\log N\rceil}{\lfloor \alpha_0 \log N\rfloor}$$

$$\geq \text{constant} \times \frac{1}{\sqrt{\log N}} \frac{N}{N^{(3\alpha_0-\epsilon)}}$$

$$\times N^{(3\alpha_0-\epsilon)\log(3\alpha_0-\epsilon)-\alpha_0\log\alpha_0-(2\alpha_0-\epsilon)\log(2\alpha_0-\epsilon)}$$

After some algebraic manipulation, we have,

$$\text{right-hand side} \geq \text{const.} \times \frac{1}{\sqrt{\log N}} N^{\alpha_0} N^{h(\epsilon)} \tag{2.35}$$

However, the left-hand side of (2.28) is equal to

$$\text{left-hand side} = t\lceil \log(t+1)\rceil = N^{\alpha_0}\lceil \alpha_0 \log N\rceil \tag{2.36}$$

Since $h(\epsilon) > 0$, a comparison of (2.35) and (2.36) shows that, for sufficiently large N, the right-hand side of (2.28) is greater than the left-hand side. This is a violation of (2.28). We have proved our assertion. □

(2.33) can be obtained in the following way: First we use (2.23) to get (assuming $t \leq N/2$),

$$T(t,N) \geq T(N^{\alpha_0}, N+N^{\alpha_0}-t) \geq T(N^{\alpha_0}, \frac{N}{2}) \tag{2.37}$$

A lower bound for $T(N^{\alpha_0}, \frac{N}{2})$ can be obtained in exactly the same way as that for $T(N^{\alpha_0}, N)$. This leads to (2.33). □

An interesting consequence of Theorem 2.12 is that, for a sorting network with N inputs, the delay time is at least $2.4 \log N$ for sufficiently large N. This result seems to be new.

2.5 Conclusions

Bounds have been given on the minimal "cost" and "delay time" of selection networks built of comparators. In particular, we have identified the leading asymptotic terms of $U(t,N)$ and $T(t,N)$ for fixed t. Many questions still remain open. For example,

(1) What is the exact value of $U(3,N)$?

(2) What is the order of magnitude of $U(\frac{N}{2},N)$ as $N \to \infty$? Does $\lim_{N\to\infty} \frac{U(\frac{N}{2},N)}{N \log N}$ exist?

(3) What is $T(\frac{N}{2},N)$?

Solutions to these problems not only are interesting in their own rights, but also will give more insight to the sorting network problem.

3. COMPUTING THE MINIMA OF QUADRATIC FORMS

3.1 Introduction

The following problem was recently raised by C. William Gear [11]: Let $F(x_1,x_2,\ldots,x_n) = \sum_{i\leq j} a'_{ij}x_ix_j + \sum_i b_ix_i + c$ be a quadratic form in n variables. We wish to compute the point $\vec{x}^{(0)} = (x_1^{(0)},\ldots,x_n^{(0)})$, at which F achieves its minimum, by a series of adaptive functional evaluations. It is clear that, by evaluating $F(\vec{x})$ at $\frac{1}{2}(n+1)(n+2)+1$ points, we can determine the coefficients a'_{ij},b_i,c and thereby find the point $\vec{x}^{(0)}$. Gear's question is, "How many evaluations are necessary?" We shall prove that $O(n^2)$ evaluations are necessary in the worst case for any such algorithm.

Assume the coefficients a'_{ij},b_i,c are such that F assumes its minimum at a unique point. Then $(x_1^{(0)},\ldots,x_n^{(0)})$ is the unique solution to the following equations:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n + b_1 = 0 \\ a_{12}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n + b_2 = 0 \\ \qquad\vdots \\ a_{1n}x_1 + a_{2n}x_2 + \ldots + a_{nn}x_n + b_n = 0 \end{cases} \tag{3.1}$$

where

$$a_{ij} = \begin{cases} 2a'_{ij} & \text{if } i = j \\ a'_{ij} & \text{if } i < j \end{cases}$$

An algorithm consists of a series of queries $F(\vec{x}^{(1)}) = ?$, $F(\vec{x}^{(2)}) = ?$, ..., $F(\vec{x}^{(m)}) = ?$, where the choice of $\vec{x}^{(j)}$ depends on the set of values $\{F(\vec{x}^{(i)}) \mid 1 \leq i \leq j-1\}$. Note that a query $F(\vec{x}) = ?$ can be written as

$$\sum_{i \leq j} s_{ij} a_{ij} + \sum_{i} t_i b_i + c = ? \qquad (3.2)$$

where the s_{ij}'s and t_i's depend on $\vec{x}$. Therefore, a lower bound for our problem can be obtained from a lower bound on the complexity of the following problem:

Problem

Solve (3.1) by means of a series of queries of the form (3.2) where each query (i.e., the choice of the numerical coefficients s_{ij}, t_i) may depend on the results of previous queries.

In this form, the problem is related to a result by Rabin [20]: It was shown there that, to solve a system of equations $\sum_{j=1}^{n} A_{ij} x_j = 0$ $(i=1,2,\ldots,n-1)$ we need $\frac{1}{2} n(n+1)-1$ tests of the form "$\vec{A}_i \cdot \vec{V} = ?$" where $\vec{A}_i = (A_{i1}, A_{i2}, \ldots, A_{in})$. The problem considered there, however, differs from the present problem in two respects:

(a) In our problem, A_{ij} and A_{ji} are not independent variables. In fact, system (3.1) corresponds to the case of having a symmetric matrix (A_{ij}).

(b) Queries of the form (3.2) are more general than the tests allowed in [20].

This latter feature makes our problem somewhat more complex than the problem studied in [20], and a different approach is required. The main result is the following:

Main Theorem

To solve the system of equations (3.1) by queries of the form (3.2), $O(n^2)$ queries are necessary.

It turns out that the present approach can also be used to rederive Rabin's result; some of the technical assumptions made in [20] can actually be removed to result in a stronger theorem.

3.2 Outline of Proof

Without loss of generality, we can assume that c = 0 in F.

(A) A few concepts. Informally, as more and more queries of the form (3.2) are asked in an algorithm, the possible values of the coefficients a_{ij}, b_i are confined to a smaller and smaller region. Correspondingly, the set of vectors $(x_1, \ldots, x_n)$ satisfying (3.1) is gradually being reduced until finally a unique point remains in the set. At that point the algorithm shall halt with the answer. Therefore, at any point in the computation, we shall denote by A, called the coefficient space, the region of those vectors (a_{ij}, b_i) which satisfy all the queries answered so far. Clearly, $A \subseteq R^p$ where $p = \frac{1}{2}(n+1)(n+2)$. Similarly, we denote by X, called the solution space (corresponding to A), the set of those vectors $(x_1, \ldots, x_n) \in R^n$ that satisfy (1) for some $(a_{ij}, b_i) \in A$. We shall also write A as $A = \bigcup_{\vec{x} \in X} A(\vec{x})$ where $A(\vec{x})$ is the linear subspace in A that corresponds to the point $\vec{x} \in X$.

Initially, A is taken to be any region in R^p such that $\det(a_{ij}) \neq 0$ for all $(a_{ij}, b_i) \in A$; and X is the solution space corresponding to this A. With each subsequent query of the form (3.2) we shall associate a query vector $E = (s_{ij}, t_i)$.

Locally, the solution space X is always an algebraic set (the set of common roots to a finite set of polynomial equations in variables

$x_1, x_2, \ldots, x_n)$ in R^n, which makes the concept of dim X well-defined. This can be proved by induction on the number of queries answered so far (cf. the proof of Lemma 3.1); details will be omitted here. We call $\vec{x}$ a simple point of X if the tangent plane to X at $\vec{x}$ has the same dimension as X itself [22]. Throughout our discussion, X will be implicitly restricted to a small neighborhood consisting of simple points only. This frees us from worrying about "bad" behavior of X. Some elementary properties of algebraic sets will be used in the sequel without being stated explicitly.

(B) Some notations. We define the functions G_i from X to A, assuming a vector $\vec{\alpha}$ is written as

$$\vec{\alpha} = (a_{11}a_{12}\ldots a_{1n}a_{22}a_{23}\ldots a_{2n}\ldots a_{n-1,n-1}a_{n-1,n}a_{nn}b_1b_2\ldots b_n) .$$

$$\begin{cases} G_1(\vec{x}) = (x_1\ x_2\ \ldots x_n\ 0\ \ 0\ \ \ldots 0\ldots\ldots \\ \qquad\qquad \ldots\ldots\ldots\ldots 0\ \ 1\ 0\ \ldots 0\ ) \\ G_2(\vec{x}) = (0\ \ x_1\ \ldots 0\ \ x_2\ x_3\ \ldots x_n \\ \qquad\qquad 0\ldots\ldots\ldots\ldots 0\ \ 0\ 1\ 0\ \ 0\ ) \\ \qquad\vdots \\ G_n(\vec{x}) = (0\ldots\ldots\ldots x_1\ldots\ldots\ldots x_2\cdot \\ \qquad\qquad \ldots\ldots\ldots x_{n-1}\ \ x_n\ 0\ldots\ldots 1\ ) \end{cases} \tag{3.3}$$

In this notation, (3.1) can be written as:

$$\vec{\alpha} \cdot G_i(\vec{x}) = 0 \qquad i=1,2,\ldots,n \tag{3.1}'$$

and a query is written as "$E \cdot \vec{\alpha} = ?$".

(C) Oracle. When a query "$E_{j+1} \cdot \vec{\alpha} = ?$" is asked, the following oracle will specify how to answer it. Later we shall show that $O(n^2)$ queries are needed by all algorithms if we give answer according to this oracle.

Let $E_1, E_2, \ldots, E_j$ be the previous query vectors, X the present solution space, and $X(c) (\subseteq X)$ the new solution space if an answer $E_{j+1} \cdot \vec{\alpha} = c$ so that the number $\max_{\vec{x} \in X(c)} \{\dim V(E_1, \ldots, E_j, G_1(\vec{x}), \ldots, G_n(\vec{x}), E_{j+1})\}$ is as large as possible, where $V(E_1, \ldots, E_j, G_1(\vec{x}), \ldots, G_n(\vec{x}), E_{j+1})$ denotes the vector space spanned by the set of vectors inside the parenthesis.

Remark We could have used an alternative oracle which simply tries to maximize $\dim X(c)$. The present formulation has essentially the same effect, but makes the analysis slightly easier.

The $j+1^{st}$ query will be called a critical query if E_{j+1} is a L.C. (linear combination) of $E_1, E_2, \ldots, E_j, G_1(\vec{x}), \ldots, G_n(\vec{x})$ for all $\vec{x} \in X$.

(D) Analysis of the oracle. Let $E_1, E_2, \ldots, E_m$ be the sequence of query vectors produced by some algorithm under the oracle given in (C). Without loss of generality, assume that they are all linearly independent. Our aim is to show that $m \geq O(n^2)$.

Lemma 3.1 Let X be the solution space when the $j+1^{st}$ query is asked and c be the answer provided by the oracle, then $\dim X(c) = \dim X$ if the $j+1^{st}$ query is not critical, and $\dim X(c) < \dim X$ if it is critical.

Proof For any $\vec{x} \in X$, $A(\vec{x})$ (as defined in Section 3.2 (A)) is the set of vectors $\vec{\alpha} \in R^p$ satisfying

$$\begin{cases} E_1 \cdot \vec{\alpha} = d_1 \\ E_2 \cdot \vec{\alpha} = d_2 \\ \quad \vdots \\ E_j \cdot \vec{\alpha} = d_j \\ G_1(\vec{x}) \cdot \vec{\alpha} = 0 \\ \quad \vdots \\ G_n(\vec{x}) \cdot \vec{\alpha} = 0 \end{cases} \tag{3.4}$$

where d_i is the answer given to the i^{th} query "$E_i \cdot \vec{\alpha} = ?$". The equations for $A(\vec{x})_c$, the new $A(\vec{x})$ after an answer $E_i \cdot \vec{\alpha} = c$ is given, are simply (3.4) plus the new equation

$$E_{j+1} \cdot \vec{\alpha} = c \tag{3.5}$$

The proof of the lemma is divided into two parts:

(i) If the $j+1^{st}$ query is not critical, it is not too difficult to show that there exists some $\vec{x} \epsilon X$, such that (a) $\vec{x} \epsilon X(c)$, (b) E_{j+1} is not a L.C. of $E_1, \ldots, E_j, G_1(\vec{x}), \ldots, G_n(\vec{x})$, and (c) $\dim V(E_1, \ldots, E_j, G_1(\vec{x}), \ldots, G_n(\vec{x}))$ is the largest possible for all $\vec{x} \epsilon X$. We wish to show that $\dim X(c) = \dim X$.

Let $\vec{\alpha} = \vec{\alpha}_0 \epsilon A$ be a solution for equations (3.4) and (3.5). It is easy to see that, for any $\vec{x}' \epsilon X$ sufficiently close to $\vec{x}$, there exists a solution (to (3.4) and (3.5)) $\vec{\alpha} = \vec{\alpha}_0'$ close to $\vec{\alpha}_0$. Therefore $X(c)$ includes all $\vec{x}' \epsilon X$ in some neighborhood of $\vec{x}$, which implies $\dim X(c) = \dim X$.

(ii) If the $j+1^{st}$ query is critical, then E_{j+1} is a L.C. of $E_1, \ldots, E_j, G_1(\vec{x}), \ldots, G_n(\vec{x})$ for all $\vec{x} \epsilon X$. This implies that the hyperplane in R^p $F \equiv \{\vec{\alpha} | E_{j+1} \cdot \vec{\alpha} = c\}$ contains $A(\vec{x})$ if $F \cap A(\vec{x}) \neq \emptyset$. Now, if

$\dim X(c) = \dim X$ is true, then there is an open set $T \subseteq S$ such that $F \cap A(\vec{x}) \neq \emptyset \quad \forall \vec{x} \epsilon T$. Consequently, F contains $\bigcup_{\vec{x}\epsilon T} A(\vec{x})$. This means that F contains A locally, which implies that E_{j+1} is a L.C. of $E_1, E_2, \ldots, E_j$, a contradiction. □

To derive another useful lemma, we define $D = \min_{\vec{x}\epsilon X}\{\dim A(\vec{x})\}$ at every stage of the computation. It is not difficult to verify (from equations (3.4) and (3.5)) that D is decreased by 1 each time a non-critical query is answered, and D does not change when a critical query is answered. As a result, the number dimA - D does not change in the former situation, and is decreased by 1 in the latter case. Since dimA - D is decreased from n to 0 in the computation process, we conclude that there must be exactly n critical queries. This, together with Lemma 3.1, implies the following:

Lemma 3.2 If the $j+1^{st}$ query is critical and $E_{j+1} \cdot \vec{\alpha} = c$ is the answer given under the oracle then $\dim X(c) = \dim X - 1$.

Proof of the Main Theorem

Let $\vec{x}^{(0)}$ be the minimum position found, and $j_1 < j_2 < \ldots < j_n$ be the query indices at which dimX is forced to be reduced. For any r ($1 \leq r \leq n$), let X be the solution space after the $j_r{}^{th}$ query is answered. According to Lemma 3.2 and Lemma 3.1, we have:

$$\text{(a)} \quad \dim X = n - r \tag{3.7}$$

$$\text{(b)} \quad \begin{cases} E_{j_1} = \text{linear combination of } E_\ell\text{'s} + \\ \qquad \text{linear combination of } G_\ell(\vec{x})\text{'s} \\ \qquad \vdots \\ E_{j_r} = \text{linear combination of } E_\ell\text{'s} + \\ \qquad \text{linear combination of } G_\ell(\vec{x})\text{'s} \\ \qquad\qquad \text{for all } \vec{x}\epsilon X \end{cases} \tag{3.8}$$

According to (3.7), there exist integers $i_1, i_2, \ldots, i_{n-r}$ such that $(x_{i_1}, x_{i_2}, \ldots, x_{i_{n-r}})$ can be used as local coordinates for X in some neighborhood of $\vec{x}^{(0)}$. Without loss of generality, we may assume these x_{i_t}'s to be $(x_{r+1}, x_{r+2}, \ldots, x_n)$.

From (3.8), it follows that for each $\vec{x} \epsilon X$, there exists integers $k_1(\vec{x}), k_2(\vec{x}), \ldots, k_r(\vec{x})$ such that $G_{k_1}(\vec{x}), G_{k_2}(\vec{x}), \ldots, G_{k_r}(\vec{x})$ are linear combinations of the E_ℓ's and the remaining $G_\ell(\vec{x})$'s. It is then easy to see that there exists a set of n-r points $Z = \{\vec{x}^{(1)}, \vec{x}^{(2)}, \ldots, \vec{x}^{(n-r)}\}$ such that:

(a) $\vec{y}^{(1)}, \vec{y}^{(2)}, \ldots, \vec{y}^{(n-r)}$ are linearly independent vectors where $\vec{y}^{(i)}$ is the projection of the vector $\vec{x}^{(i)}$ on its last n-r components. (3.9)

(b) There exists integers $1 \leq i_1 < i_2 < \ldots < i_r \leq n$ such that, for all $\vec{x} \epsilon Z$, $G_{i_1}(\vec{x}), G_{i_2}(\vec{x}), \ldots, G_{i_r}(\vec{x})$ are linear combinations of E_ℓ's and the remaining $G_\ell(\vec{x})$'s. (3.10)

Now, let V_1 be the linear space spanned by $\{G_{i_1}(\vec{x}), \ldots, G_{i_r}(\vec{x}) \mid \vec{x} \epsilon Z\}$, and V_2 be the linear space spanned by the E_ℓ's and the other $G_\ell(\vec{x})$'s ($\ell \neq i_t \ \forall t$). It is clear by (3.10) that

$$\dim V_2 \geq \dim V_1 \tag{3.11}$$

and

$$m+(n-r)^2 \geq \dim V_2 \tag{3.12}$$

Furthermore, the following proposition is true:

Proposition $$\dim V_1 \geq (n-r)(r-(n-r)) \tag{3.13}$$

Proof of Proposition Obviously $i_1, i_2, \ldots, i_{r-(n-r)} < r+1$. By explicitly examining the forms of $G_i(\vec{x})$'s in equation (3.3) and making use of (3.9),

it is seen that all the vectors in the set $Q = \{G_{i_t}(\vec{x}) \mid 1 \leq t \leq r-(n-r), \vec{x} \in Z\}$ are linearly independent. It follows that

$$\dim V_1 \geq |Q| = (n-r)(r-(n-r))$$

which is (3.13). □

Equations (3.11), (3.12), and (3.13) lead to

$$m+(n-r)^2 \geq (n-r)(r-(n-r))$$

Taking $r = \frac{5}{6} n$, we obtain

$$m \geq \frac{1}{12} n^2 \tag{3.14}$$

Thus, at least $O(n^2)$ queries are needed. This completes the proof of our main theorem. □

3.3 Concluding Remarks

We have shown that $O(n^2)$ adaptive functional evaluations are required to find the minimum point of a quadratic form. It seems likely that an exact bound could be obtained by making more effective use of the lemmas. As mentioned earlier, our approach can also be used to prove a stronger version of the result derived in [20].

4. FINDING MINIMUM SPANNING TREES

4.1 Introduction

Given a connected, undirected graph $G = (V,E)$ and a function c which assigns a cost $c(e)$ to every edge $e \in E$, it is desired to find a spanning tree T for G such that $\sum_{e \in T} c(e)$ is minimal. In this note we describe an algorithm which finds a minimum spanning tree (MST) in $O(|E| \log \log |V|)$ time. Previously the best MST algorithms known have running time $O(|E| \log |V|)$ for sparse graphs [1]; and more recently Tarjan [23] has an algorithm that requires $O(|E| \sqrt{\log |V|})$ time.

Our algorithm is a modification of an algorithm by Sollin [2]. His method works by successively enlarging components of the MST. In the first stage the minimum-cost edge incident upon each node of G is found. These edges are part of the MST sought. The groups of vertices that are connected by these edges are then identified. By shrinking each such group of vertices to a single node, we obtain a new graph with at most $|V|/2$ nodes. This process is repeated for a number of times, at each stage for a new graph, until finally a single contracted node remains. Clearly each stage of this procedure involves $O(|E|)$ operations, and $\log |V|$ stages are necessary in the worst case. Thus this algorithm requires a total of $O(|E| \log |V|)$ operations.

In our algorithm, we first partition the set of edges incident with each node v into k levels $E_v^{(1)}$, $E_v^{(2)}$, ..., $E_v^{(k)}$ so that $c(e) \leq c(e')$ if $e \in E_v^{(i)}$, $e' \in E_v^{(j)}$ and $i < j$. This can be done in $O(|E| \log k)$ time by

repeatedly applying the linear median-finding algorithm [3]. Having accomplished this, we follow basically Sollin's algorithm as outlined above. Note that the number of operations needed in this phase is now reduced to $O(\frac{|E|}{k}\log|V|)$ since only approximately $|E|/k$ edges have to be examined at each stage to find the minimum-cost edges incident with all the nodes. Therefore, the total number of operations required by our algorithm is $O(|E|\log k + \frac{|E|}{k}\log|V|)$, which is $O(|E|\log\log|V|)$ if we choose k to be $\log|V|$.

4.2 Algorithm

For the moment, assume $|E| \geq |V|\log|V|$. If $|E| < |V|\log|V|$, the algorithm needs a slight modification as will be discussed later.

The algorithm uses three sets T, VS, and ES. T is used to collect edges of the final spanning tree. The set VS contains the vertex sets corresponding to the connected components of the spanning tree found so far. And ES contains, for each vertex set W in VS, an edge set E(W). Initially we have VS = $\{\{v\}|v \in V\}$ and ES = {{all the edges incident upon v}$|v \in V\}$. The algorithm also uses an integer parameter k, a level function ℓ: V → {1,2,...,k,k+1}, and a function low: V → real numbers.

```
Procedure MST;
   begin
      T ← ∅;  VS ← ∅;  ES ← ∅;
      for each vertex v ∈V do
         begin add the singleton set {v} to VS;
               add the set E({v}) = {all the edges incident with v} to ES;
A.             divide E({v}) into k levels of equal size according to
                  cost, i.e., obtain E_v^(1),E_v^(2),...,E_v^(k) with the property
                  that ⋃_{j=1}^{k} E_v^(j) = E({v}) and c(e) ≦c(e') if e ∈E_v^(i),e'∈E_v^(j),
                  and i < j;
               set ℓ(v) ← 1;
         end
      while |VS| > 1 do
         begin
B.             take a vertex set W from VS;
               for each vertex v ∈W do
                  begin low(v) ← ∞;
                     while low(v) = ∞ and ℓ(v) ≦ k do
                        begin for each edge e = (v,v') in E_v^(ℓ(v)) do
                                 begin
C.                                  if v'∈W then delete e from E_v^(ℓ(v));
D.                                  else low(v) ← min{low(v), c(e)};
                                 end
                              if low(v) = ∞ then ℓ(v) + 1;
                        end
                  end
F.             find the edge e = (v,v') in E(W) whose cost is equal to
                  min{low(v) |v ∈W};
H.             in VS, replace W and the vertex set W' containing v' by
                  W ∪W';
I.             in ES, replace E(W) and E(W') by E(W) ∪ E(W');
               add e to T;
         end
      output T;
   end MST
```

4.3 Remarks

(1) In the above procedure, the set VS is implemented with a circular queue. Step B corresponds to removing W from the front of the queue; step H corresponds to deleting W' and adding the new W to the tail of the queue. A full cycle of the queue, in which every vertex set of VS is merged with some others, corresponds to one "stage" of the algorithm as discussed before.

(2) Step A is done by applying the median-finding algorithm [3] repeatedly, and takes $O(|E|\log k)$ time.

(3) Step C is executed at most $2|E|$ times, since each edge (v,v') of G can be thrown out at most twise--once as (v,v'), once as (v',v).

(4) Steps D and F amount to approximately

$$\log|V| \sum_{v\in V} \left\lceil \frac{|E(\{v\})|}{k} \right\rceil \leq \log|V| \left(\frac{2|E|}{k} + |V|\right)$$

min operations.

(5) The set union operation in step I can be implemented in a straightforward manner; for example, as in [21]. The total number of operations incurred is $O(|V|\log|V|)$.

It follows that the total cost of this algorithm is of the order

$$|E|\log k + \log|V| \frac{|E|}{k} + |V|\log|V| \ . \qquad (4.1)$$

Taking $k = \log|V|$ and noting $|E| \geq |V|\log|V|$, the above expression is bounded by const. $\times$ $(|E|\log\log|V|)$.

If $|E| < |V|\log|V|$ for a graph G, we will first let $k = 1$ and execute procedure MST until each vertex set in VS is of size at least $\log|V|$. This process takes at most $\log\log|V|$ "stages" since the size of the smallest vertex set in VS at least doubles after each stage. Hence the amount of work involved in this process is $O(|E|\log\log|V|)$. The result can be regarded as a new graph $G' = (V',E')$ where $|V'| \leq |V|/\log|V|$ and $|E'| \leq |E|$. We now apply Procedure MST to G' with $k = \log|V|$. The number of operations required is given by (4.1),

$$|E'|\log k + \log|V'|\,\frac{|E'|}{k} + |V'|\log|V'|$$

$$\leq |E|\log k + \log|V|\,\frac{|E|}{k} + |V|,$$

which is again $O(|E|\log\log|V|)$ for $k = \log|V|$.

5. SCHEDULING UNIT-TIME TASKS WITH LIMITED RESOURCES

5.1 Introduction

One great advantage offered by the multiprocessing system is the potential decrease in computation time. Because of the sequential constraints that often exist among the tasks, a good scheduling algorithm is essential for the efficient utilization of parallel processing facilities.

Various aspects of an abstract multiprocessor model have been studied in the literature [4], [5], [6], [8], [13], [19]. That model considers a set of tasks $\{T_1, T_2, \ldots, T_r\}$ to be processed on n identical processors. A partial order $<\cdot$ on $\{T_1, \ldots, T_r\}$ is given, and a function $\mu(T_i)$ which determines the execution time for T_i is specified. In a schedule a task cannot be executed unless all of its predecessors (in $<\cdot$) have been completed. Recently Garey and Graham [9] incorporated the idea of "resources" into this model. Each task requires certain amounts of resources for its execution. A schedule now has to satisfy the additional constraint that the total demand for each resource cannot exceed a fixed amount at any instant. It was found [9] that, in general, the efficiency of a schedule derived according to an arbitrary priority list is completely unpredictable.

We examine an interesting case of the Garey-Graham model in which each task takes unit time to complete. Since efficient methods for finding optimal schedules seem unlikely to exist even without the

resource complication [23], it is important to study practical heuristic algorithms. In the present paper, the worst-case behavior of several heuristic algorithms is analyzed. In particular, the total execution time resulting from the use of certain heuristic algorithms is shown to differ from the best possible by no more than a multiplying factor that depends on the number of different resources only.

Some results derived here can be generalized to the case when the lengths of execution time for tasks are not the same. This will be briefly discussed in section 10. Even with the "unit-time" restriction, however, this model is of some practical interest in view of its close connection with preemption scheduling [6].

Main results are stated in sections 4 and 5. The proofs are given in sections 6, 7, 8, and 9.

5.2 The Model

A system consists of $(n, s, F, <\cdot, \vec{R})$ where n and s are positive integers, $F = \{T(1), T(2), \ldots, T(r)\}$ a set of tasks, $<\cdot$ a partial order on F, and $\vec{R}$ is a vector function with s components defined on F. We require $0 \leq R_j(T(i)) \leq 1 \ \forall i,j$ where $\vec{R}(T(i)) \equiv (R_1(T(i)), R_2(T(i)), \ldots, R_s(T(i)))$.

(A) A schedule f is a finite sequence of non-empty subsets of tasks $F_1, F_2, \ldots, F_\omega$ such that:

(i) $\bigcup_{i=1}^{\omega} F_i = F$ and $F_i \cap F_j = \emptyset \quad \forall i \neq j$

(ii) if $T(i) <\cdot T(j)$, $T(i) \in F_k$, $T(j) \in F_\ell$ then $k < \ell$

(iii) $|F_k| \leq n \quad \forall k$

$$\text{(iv)} \quad \sum_{T(i)\in F_k} R_\ell(T(i)) \leq 1 \qquad \forall k, \ell$$

ω is said to be the running time of schedule f.

(B) Interpretation. n is the number of processors, s the number of resources, $R_j(T(i))$ the amount of resource j demanded by the execution of task i, and F_k is the set of tasks executed simultaneously between time k-1 and k.

(C) List schedule. Given a list (i.e., a permutation of the r tasks), $L = (T(i_1), T(i_2), \ldots, T(i_r))$, a schedule f is generated as follows:

Step (a) Set $i \leftarrow 1$.

Step (b) Let $F_i \leftarrow \emptyset$.

Step (c) If list is empty, stop. Else scan list from the beginning; find the first task $T(i_j)$ such that if we let $F_i \leftarrow F_i \cup \{T(i_j)\}$, all the conditions (ii), (iii) and (iv) of (A) are not violated; set $F_i \leftarrow F_i \cup \{T(i_j)\}$ and delete $T(i_j)$ from the list.

Step (d) If $|F_i| = n$ or no eligible $T(i_j)$ could be found in step (c), then set $i \leftarrow i + 1$ and go to step (b). Otherwise goto (c).

(D) Some useful definitions.

Definition 5.1 m and W are functions defined on F by:

$$m(T(i)) = \max\{R_j(T(i)) \mid 1 \leq j \leq s\}$$

$$W(T(i)) = \sum_{j=1}^{s} R_j(T(i))$$

Definition 5.2 Let A be a set of tasks. $R_j(A)$, m(A), W(A) are defined by:

$$R_j(A) = \sum_{T(i)\in A} R_j(T(i))$$

$$m(A) = \max_j \{R_j(A)\}$$

$$W(A) = \sum_{j=1}^{s} R_j(A)$$

5.3 Algorithms to be Considered

The following algorithms are used to generate lists, which in turn produce schedules as was described in section 2.

(A) Arbitrary list. Just form any list.

(B) Level algorithm. This algorithm and its variants have been considered by several authors [2], [5], [6]. First a level function H is defined by:

H(T(i)) = M if the longest chain of tasks that starts with T(i) has length M.

A list L is then defined by the following linear order relation α (T(i) appears before T(j) in L if T(i) α T(j)).

(i) T(i) α (j) if H(t(i)) > H(T(j))

(ii) Let T(i) α T(j) if H(T(i)) = H(T(j)) and i < j.

(C) Resource decreasing algorithm. This is a generalization of the first fit decreasing algorithm used in bin packing problem [10]. A linear order α for L is defined by:

(i) T(i) α (Tj) if m(T(i)) > 1/2 and m(T(j)) $\leq$ 1/2.

(ii) If (i) is not applicable, then T(i) α T(j) if W(T(i)) > W(T(j)).

(iii) In (ii), if W(T(i)) = W(T(j)), then T(i) α T(j) if i < j.

5.4 Bounds on the Worst-Case Behavior

We shall discuss the worst-case behavior of algorithms defined in the last section. As explained in section 2, n denotes the number of processors, r the number of tasks, s the number of different resources, and $<\cdot$ a partial order on the set of tasks. Let ω_0 be the running time for an optimal schedule (i.e., one which requires minimal running time), and ω_L be the running time associated with list L.

Theorem 5.3 If $n \geq r$, then

$$\frac{\omega_N}{\omega_0} \leq \frac{1}{2} s(\omega_0 + 7s) \qquad \text{for any list L} \tag{5.1}$$

The following theorem states that (5.1) can be essentially achieved.

Theorem 5.4 There exists systems with arbitrary large ω_0 and $n \geq r$ for which

$$\frac{1}{2} s(\omega_0 - 2s) \leq \frac{\omega_L}{\omega_0} \qquad \text{for some list L} \tag{5.2}$$

When the condition $n \geq r$ is dropped, an upper bound for $\frac{\omega_L}{\omega_0}$ can readily be obtained if we regard "processors" also as a resource. However, a more detailed analysis yields the following stronger result:

Theorem 5.5 For any n and any list L,

$$\frac{\omega_L}{\omega_0} \leq \frac{n-1}{2n} s\omega_0 + \frac{7(n-1)}{2n} s + 1 \tag{5.3}$$

Formula (5.2) shows that, unlike in the conventional model where "resources" are not taken into account, the worst-case behavior of ω_L/ω_0 is not bounded by a constant. The following theorems show that this behavior improves drastically if some efforts are made in preparing the list.

Theorem 5.6 If a list L is prepared by using level algorithm, then:

$$\frac{\omega_L}{\omega_0} \leq \frac{n-1}{n}(2s+1) + 1 \qquad \text{for arbitrary } n \tag{5.4}$$

$$\frac{\omega_L}{\omega_0} \leq 2s + 1 \qquad \text{if } r \leq n \tag{5.5}$$

Theorem 5.7 For any given $\epsilon > 0$, there exists systems with arbitrary large ω_0 such that

$$\frac{17}{10}s - \epsilon < \frac{\omega_L}{\omega_0} \qquad \text{where L is generated by level algorithm} \tag{5.6}$$

If the resource decreasing algorithm described in section 3 is used, a still better upper bound exists for ω_L/ω_0.

Theorem 5.8 If a list L is obtained by using resource decreasing algorithm, then:

$$\frac{\omega_L}{\omega_0} \leq \frac{n-1}{n}\left(\frac{7}{4}s+1\right) + 1 \qquad \text{for any } n \tag{5.7}$$

$$\frac{\omega_L}{\omega_0} \leq \frac{7}{4}s + 1 \qquad \text{if } r \leq n \tag{5.8}$$

An asymptotic lower bound is given by:

Theorem 5.9 For any given $\epsilon > 0$, there exists systems with arbitrary large ω_0 such that

$$\frac{3}{2} s - \epsilon < \frac{\omega_L}{\omega_0} \qquad \text{where L is generated by resource decreasing algorithm} \tag{5.9}$$

5.5 The Special Case in which $<\cdot$ is Empty

An upper bound for ω_L/ω_0 was derived in [9] under the assumption $r \leq n$ for this special case. That bound $\omega_L/\omega_0 \leq s+1$ is true even when the lengths of execution time for tasks are not the same. The following theorem gives a slightly better asymptotic upper bound when uniform execution is assumed.

Theorem 5.10 If $r \leq n$, and $<\cdot$ empty, then for any given $\epsilon > 0$ and an arbitrary list L,

$$\frac{\omega_L}{\omega_0} \leq s + \frac{17}{20} + \epsilon \qquad \text{if } \omega_0 \text{ is large enough} \tag{5.10}$$

The following theorem can be obtained by constructing examples.

Theorem 5.11 For any given $\epsilon > 0$, then exists systems with $<\cdot$ empty, and arbitrary large ω_0 satisfying

$$s + \frac{7}{10} - \epsilon < \frac{\omega_L}{\omega_0} \qquad \text{for some list L} \tag{5.11}$$

Theorem 5.12 If $<\cdot$ is empty, then for any $e > 0$, and any list L,

$$\frac{\omega_L}{\omega_0} \leq \frac{n-1}{n}\left(s + \frac{17}{20}\right) + 1 + \epsilon \qquad \text{for large } \omega_0 \tag{5.12}$$

(5.10) and (5.12) may not be the best possible bounds. For example, a stronger statement can be proved for the case $s = 2$.

Theorem 5.13 If $s = 2$, $r \leq n$, and $<\cdot$ is empty, then for any given $\epsilon > 0$,

$$\frac{\omega_L}{\omega_0} \leq \frac{41}{15} + \epsilon \quad \text{for large } \omega_0 \tag{5.13}$$

It is interesting to note that the scheduling problem with $s = 1$, $r \leq n$, and $<\cdot$ empty is equivalent to the bin-packing problem. The resource demanded by a task corresponds to the weight of an object, and the running time ω corresponds to the number of boxes used in the bin-packing problem. If we extend the concept of weight to an s-dimensional weight-vector, then our problem considered here can be regarded as a bin-packing problem with multi-weight. We shall return to this point in section 9.

5.6 Proof of Theorems 5.3, 5.4, and 5.5

(A) Proof of Theorem 5.3. Let f be the schedule generated by list L, and $F_1, F_2, \ldots, F_\omega$ be the sets of tasks corresponding to f as defined in section 2. Several preliminary results are needed to establish the theorem.

Lemma 5.14 For any task $T(j_1)$, there exist a sequence of integers $a_q < a_{q-1} < \ldots < a_2 < a_1$ and a chain of tasks $T(j_q) <\cdot T(j_{q-1}) <\cdot \ldots <\cdot T(j_2) <\cdot T(j_1)$ such that

(i) $T(j_\ell) \in F_{a_\ell}$ for $\ell = 1, 2, \ldots, q$.

(ii) $m(T(j_q)) > 1/2$, unless for all $\ell (1 \leq \ell < a_q)$, $m(F_\ell) > 1/2$.

(iii) If $\ell \neq a_i \; \forall i$, and $a_q < \ell < a_1$, then $m(F_\ell) > 1/2$.

Proof The following procedure generates a sequence of integers $a_1, a_2, \ldots, a_q$ and a chain of tasks $T(j_1)$, $T(j_2)$, ..., $T(j_q)$ that satisfy the required conditions.

Step (a) (Initialization) Let a_1 be such that $T(j_1) \in F_{a_1}$;
$t \leftarrow 1$; $v \leftarrow a_1$.

Step (b) (Termination condition) If $m(T(j_t)) > 1/2$ or $v = 1$
then stop.

Step (c) (Try to find a $T(j_{t+1}) \in F_{v-1}$ such that $T(j_{t+1}) <\cdot T(j_t)$.
This is always possible if $m(F_{v-1}) \leq 1/2$, since $T(j_t)$
could not be executed between time v-2 and v-1.)
If $\exists$ a u such that $T(u) <\cdot T(j_t)$ and $T(u) \in F_{v-1}$,
then $v \leftarrow v-1$, $t \leftarrow t+1$, $j_t \leftarrow u$, $a_t \leftarrow v$
else $v \leftarrow v-1$;

Step (d) goto Step (b).

Condition (ii) stated in the lemma is satisfied by the termination condition in Step (b). □

Lemma 5.15 For some k, there exist k disjoint chains $C_1, C_2, \ldots, C_k$ of tasks such that the following is true:

Let C_i be given by $T(j_{iq_i}) <\cdot T(j_{i,q_i-1}) <\cdot \ldots <\cdot T(j_{i1})$, then

(i) $m(T(j_{iq_i})) > 1/2$ for i=1,2,...,k-1 (5.14)

(ii) If F_ℓ is such that $F_\ell \cap C_i = \emptyset$ for i=1,2,...,k, then $m(F_\ell) > 1/2$. (5.15)

Proof Choose an arbitrary task $T(j_{11}) \in F_\omega$. Find a chain C_1 using the procedure described in the proof of Lemma 5.14. Let this chain be $T(j_{1q_1}) <\cdot T(j_{1,q_1-1}) <\cdot \ldots <\cdot T(j_{11})$. Suppose $T(j_{1q_1}) \in F_\ell$. If $m(F(j_{1q_1})) \leq 1/2$ or $\ell = 1$, then stop. Otherwise, choose an arbitrary task $T(j_{21}) \in F_{\ell-1}$. We then construct a new chain C_2 according to the

procedure in the proof of Lemma 5.14. Let C_2 be $T(j_{2q_2}) <\cdot T(j_{2,q_2-1}) <\cdot \ldots <\cdot T(j_{21})$. Again we check if $m(T(j_{2q_2})) \leq 1/2$ or $T(j_{2q_2}) \in F_1$, and decide whether to construct C_3. This process is repeated until $m(T(j_{kq_k})) \leq 1/2$ or $T(j_{kq_k}) \in F_1$ where $T(j_{kq_k})$ is the maximum element of the last chain obtained. At this point, k chains $C_1, C_2, \ldots, C_k$ with respective lengths $q_1, q_2, \ldots, q_k$ have been obtained. It is straightforward to verify that they satisfy condition (ii) by using Lemma 5.14. Figure 5.1 illustrates the result of this process. □

Lemma 5.16 Let $\{j_{ih} | 1 \leq i \leq k,\ 1 \leq h \leq q_i\}$ be the same as in Lemma 5.15. There exists an integer d $(1 \leq d \leq s)$ and a set of integers $V \subset \{1,2,\ldots,k-1\}$ such that

$$\text{(i)} \quad R_d(T(j_{iq_i})) > 1/2 \quad \text{for } i \in V \tag{5.16}$$

$$\text{(ii)} \quad \sum_{i \in V} q_i + q_k \geq \frac{\sum_{i=1}^{k} q_i}{s} \tag{5.17}$$

Proof Define s sets of integers by

$$V_\ell \equiv \{i \mid 1 \leq i \leq k-1,\ R_\ell(T(j_{iq_i})) > 1/2\}$$

$$\ell = 1,2,\ldots,s \ .$$

Since $\bigcup_\ell V_\ell = \{1,2,\ldots,k-1\}$, we have

$$\sum_{\ell=1}^{s} \left(\sum_{i \in V_\ell} q_i \right) \geq \sum_{i=1}^{k-1} q_i$$

Therefore, there exists d such that

$$\sum_{i \in V_d} q_i \geq \frac{1}{s} \sum_{i=1}^{k-1} q_i$$

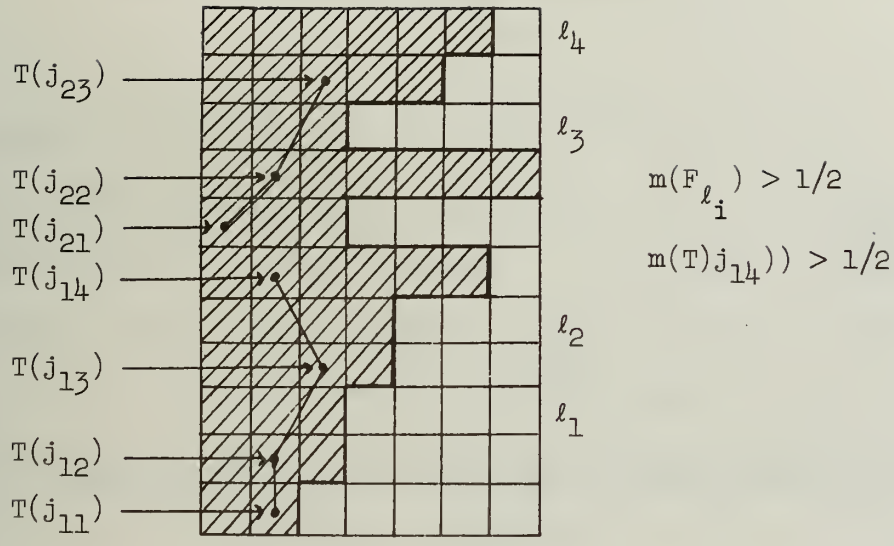


Figure 5.1 Disjoint chains $C_1, C_2, \ldots, C_k$.

Let $V \equiv V_d$, we have thus

$$\sum_{i \epsilon V} q_i + q_k \geq \frac{1}{s} \sum_{i=1}^{k-1} q_i + q_k \geq \frac{1}{s} \sum_{i=1}^{k} q_i$$

□

We are now ready to prove Theorem 5.3. Let

$$G \equiv \{\ell \mid F_\ell \cap (\bigcup_{i=1}^{k} C_i) = \emptyset\}$$

Obviously

$$\sum_{i=1}^{k} q_i + |G| = \omega$$

Since the total amount of resource available at any instant is s, we have

$$\omega_0 \geq \frac{\sum_i W(T_i)}{s} \geq \frac{\sum_{\ell \epsilon G} W(F_\ell) + \sum_{i=1}^{k-1} W(T(j_i, q_i))}{s}$$

$$\geq \frac{\sum_{\ell \epsilon G} m(F_\ell) + \sum_{i=1}^{k-1} m(T(j_i, q_i))}{s}$$

By virtue of (14) and (15), we have

$$\omega_0 \geq \frac{\frac{1}{2}|G| + \frac{1}{2}(k-1)}{s}$$

Thus,

$$2s\omega_0 \geq |G| + k-1 \qquad (5.19)$$

Now, according to Lemma 5.16, there exists a set V such that (5.16) and (5.17) are true. Let us renumber the chains C_i's ($i \in V$) as $C'_1, C'_2, \ldots, C'_{k_1}$ where $k_1 \equiv |V|$. Let the length of C'_i be q'_i and let its maximum task be $T(t_i)$. It follows from (5.17) that

$$\sum_{i=1}^{k_1} q'_i + q_k \geq \frac{1}{s} \sum_{i=1}^{k} q_i \tag{5.20}$$

Furthermore, no two tasks in the set $\{T(t_i) | i=1,2,\ldots,k_1\}$ can be executed simultaneously in any schedule since each of them demands more than 1/2 unit of the d^{th} resource. In the optimal schedule, we can assume, without loss of generality, that $T(t_i)$ is done before $T(t_{i+1})$ for $1 \leq i \leq k_1-1$. Using the starting time of $T(t_1)$ as a reference point in time, the last task in chain C'_i cannot be done in less than $q'_i + (i-1)$ time units. Therefore,

$$\begin{aligned}
\omega_0 &\geq \max(q_k, q'_1, q'_2+1, q'_3+2, \ldots, q'_{k_1}+(k_1-1)) \\
&\geq \frac{1}{k_1+1}(q_k+q'_1+(q'_2+1)+\ldots+(q'_{k_1}+(k'_1-1))) \\
&= \frac{1}{k_1+1}(q_k + \sum_{i=1}^{k-1} q'_i + \frac{1}{2} k_1(k_1-1)) \\
&\geq \frac{1}{k_1+1}(\frac{1}{s} \sum_{i=1}^{k} q_i + \frac{1}{2} k_1(k_1-1))
\end{aligned} \tag{5.21}$$

where (5.20) is used in the last step. (5.21) can be written as

$$s(k_1+1)\omega_0 \geq \sum_{i=1}^{k} q_i + \frac{1}{2} sk_1(k_1-1) \tag{5.22}$$

From (5.18), (5.19), and (5.22), it is straightforward to deduce that

$$-\frac{1}{2}sk_1(k_1-1) + s(k_1+1)\omega_0 + 2s\omega_0 \geq \omega + k - 1 \geq \omega \tag{5.23}$$

(5.23) can be written as

$$-\frac{1}{2}s(k_1-(\omega_0+\frac{1}{2}))^2 + 3s\omega_0 + \frac{1}{2}s(\omega_0+\frac{1}{2})^2 \geq \omega \tag{5.24}$$

Now, since ω_0, k_1 are integers, $|k_1-(\omega_0+\frac{1}{2})| \geq 1/2$. Thus (5.24) implies

$$-\frac{1}{2}s(\frac{1}{2})^2 + 3s\omega_0 + \frac{1}{2}s(\omega_0+\frac{1}{2})^2 \geq \omega$$

That is,

$$\frac{1}{2}s\omega_0^2 + \frac{7}{2}s\omega_0 \geq \omega \tag{5.25}$$

This completes the proof of Theorem 5.3. □

(B) Proof of Theorem 5.4. We need only to exhibit a system with a schedule L such that $\frac{\omega_L}{\omega_0} > \frac{1}{2}s\omega_0 - s^2$.

Let $F = \{T^{(k)}, T^{(k)}_{j\ell} \mid 1 \leq k \leq s,\ 1 \leq j \leq q,\ 1 \leq \ell \leq j\}$. The partial order $<\cdot$ is represented as a precedence graph in Figure 5.2.

$\vec{R}$ is defined by:

$$\vec{R}(T^{(k)}) = (1,1,\ldots,1) \qquad k = 1,2,\ldots,s$$

$$\vec{R}(T^{(k)}_{il}) = (0,0,\ldots,0,1-\frac{\delta}{2^{i-1}},0,\ldots 0)$$

$$k = 1,2,\ldots,s;\ i = 1,2,\ldots,q$$

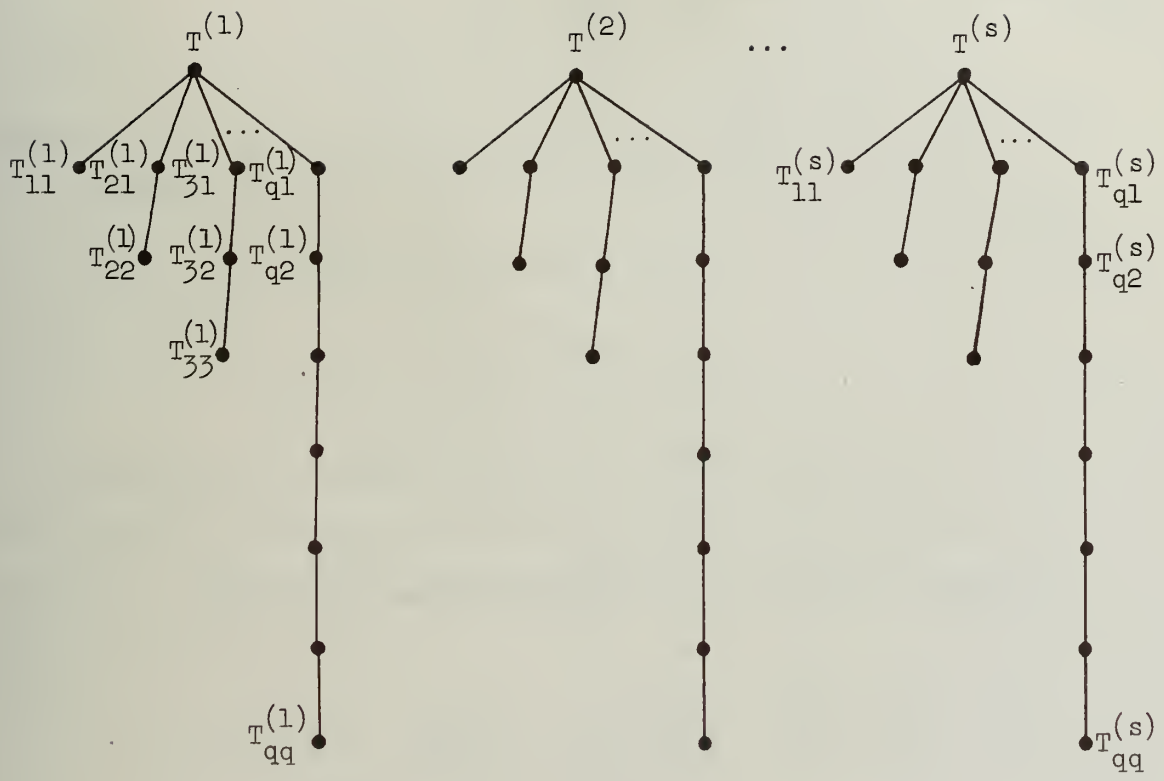


Figure 5.2 A "bad" partial order for the arbitrary list heuristics.

$$\vec{R}(T_{ij}^{(k)}) = (0,0,\ldots,0,\frac{\delta}{2^{i-1}},0,\ldots,0)$$

$$k = 1,2,\ldots,s;\ i = 1,2,\ldots,q;$$

$$j = 1,2,\ldots,i$$

where δ is a small positive number.

Let $L \equiv L_1 L_2 \ldots L_s$ where, for $k=1,2,\ldots,s$, $L_k = (T^{(k)}, T_{11}^{(k)}, T_{21}^{(k)}, T_{22}^{(k)}, \ldots, T_{q1}^{(k)}, T_{q2}^{(k)}, \ldots, T_{qq}^{(k)})$. For this list L, only one task is executed at any time. The total execution time is, therefore, equal to $s(1+1+2+3+\ldots+q)$. Thus,

$$\omega_L = s(1+\frac{1}{2}q(q+1)).$$

On the other hand, the optimal schedule is generated by

$$L_0 \equiv (T^{(1)}, T^{(2)}, \ldots, T^{(s)}) L_1' L_2' \ldots L_s' \text{ where, for each K,}$$

$$L_k' = (T_{q1}^{(k)}, T_{q2}^{(k)}, \ldots, T_{qq}^{(k)}, T_{q-1,1}^{(k)}, T_{q-1,2}^{(k)}, \ldots, T_{q-1,q-1}^{(k)}, \ldots,$$

$$T_{21}^{(k)}, T_{22}^{(k)}, T_{11}^{(k)}) .$$

In this case $\omega_0 = q + s$.

Thus

$$\frac{\omega_L}{\omega_0} = \frac{s(1+\frac{1}{2}q(q+1))}{q+s} > \frac{1}{2}s\omega_0 - s^2$$

□

(C) Proof of Theorem 5.5. The proof goes essentially the same as the proof for Theorem 4.1. The only difference is that, if $\ell \in G$ where $G \equiv \{\ell \mid F_\ell \cap (\bigcup_{i=1}^{k} C_i) = \emptyset\}$, then either $m(F_\ell) > 1/2$ or $|F_\ell| = n$. Let

$$Q \equiv \{\ell \mid \ell \epsilon G,\ m(F_\ell) > 1/2\}$$

and $Q' \equiv G - Q$. Then equation (5.18) is modified to

$$\sum_{i=1}^{k} q_i + |Q| + |Q'| = \omega \tag{5.26}$$

while equations (5.19) and (5.22) are unchanged,

$$2s\omega_0 \geq |Q| + k-1 \geq |Q| \tag{5.27}$$

$$s(k_1+1)\omega_0 \geq \sum_{i=1}^{k} q_i + \frac{1}{2} sk_1(k_1-1) \tag{5.28}$$

A new constraint arises from the fact that there are now only n processors. Since there are at least $\sum_{i=1}^{k} q_i + |Q| + n|Q'|$ tasks, we have

$$n\omega_0 \geq \sum_{i=1}^{k} q_i + |Q| + |Q'| \tag{5.29}$$

Eliminating $|Q|$, $|Q'|$, and $\sum_{i=1}^{k} q_i$ from (5.26), (5.27), (5.28), and (5.29), we obtain:

$$\omega \leq \frac{n-1}{n}(-\frac{1}{2} sk_1(k_1-1)+(k_1+1)s\omega_0 + 2s\omega_0) + \omega_0 \tag{5.30}$$

The following equation can then be obtained from (5.3) in the same manner that (5.25) is obtained from (5.23).

$$\omega \leq \frac{n-1}{n}(\frac{1}{2} s\omega_0^{\,2} + \frac{7}{2}s\omega_0) + \omega_0 \tag{5.31}$$

5.7 Proof of Theorems 5.6 and 5.7

(A) Proof of Theorem 5.6. We shall only prove equation (5.5). The other inequality, equation (5.4), can be similarly obtained (cf. the argument used in section 6 (C)).

Consider a schedule generated by the level algorithm. Let ω be the completion time and $F_1, F_2, \ldots, F_\omega$ be defined as in section 2. We introduce a function u defined as follows:

$$u(\ell) \equiv \max_{T(i)\in F_\ell} \{H(T(i))\} \quad \text{for } \ell=1,2,\ldots,\omega \tag{5.32}$$

where H is the level function defined in section 2.

Lemma 5.17 $u(\ell) \geq u(\ell+1)$ for $\ell=1,2,\ldots,\omega-1$

Proof Suppose otherwise. Then there exists an ℓ for which $u(\ell) < u(\ell+1)$. Now, let $T(j)\in F_{\ell+1}$ be a task with $H(T(j)) = u(\ell+1)$. Then $H(T(j)) > H(T(i))$ for all $T(i)\in F$. Thus, T(j) must appear before all $T(i)\in F_\ell$ in the list since it is prepared by the level algorithm. Furthermore, there is no $T_i \in F_\ell$ for which $T_i <\cdot T_j$. Therefore, T_j should be done no later than any of the task in F_ℓ. This contradicts the assumption that $T_j \in F_{\ell+1}$. □

Lemma 5.18 If $U(\ell) = u(\ell+1)$, then $W(F_\ell) + W(F_{\ell+1}) > 1$.

Proof Similar to the proof for Lemma 5.17. □

Now let $A \equiv \{\ell \mid u(\ell) = u(\ell+1)\}$, then

$$2 \sum_{i=1}^{\omega} W(F_i) \geq \sum_{\ell\in A} (W(F_\ell) + W(F_{\ell+1})) > |A| \tag{5.33}$$

Thus,

$$\omega_0 \geq \frac{1}{s} \sum_{i=1}^{\omega} W(F_i) > \frac{1}{2s} |A| \tag{5.34}$$

On the other hand, according to Lemma 5.17, we have:

$$u(\ell) \geq u(\ell+1) + 1 \quad \text{for } \ell \notin A,\ 1 \leq \ell \leq \omega-1$$

Therefore,

$$\omega_0 \geq u(1) \geq \omega - |A| \tag{5.35}$$

(5.34) and (5.35) imply $(2s+1)\omega_0 \geq \omega$. This proves the theorem. □

(B) Proof of Theorem 5.7. Consider a system of tasks with partial order $<\cdot$ as is shown in Figure 5.3.

For all $1 \leq i \leq s$, $1 \leq k \leq q$, let

$$\vec{R}(T_i) = (1,1,\ldots,1),\ \vec{R}(T_i') = (\vec{R}(T_i'') = (0,0,\ldots,0)$$

$$R_1(T_{1k}) = R_2(T_{2k}) = \ldots = R_s(T_{sk}) \neq 0$$

$$R_j(T_{ik}) = 0 \quad \text{if} \quad j \neq i$$

The values of $R_i(T_{ik})$ are unspecified at this point.

The list generated by the level algorithm is $L = L_1L_2\ldots L_s$ where

$$L_i = (T_iT_i'T_i''T_{i1}T_{i2}\cdots T_{iq}) \quad i=1,2,\ldots,s.$$

It is easy to see that T_{i+1} and T_{i+1}' are done after the set of tasks $\{T_{i1},T_{i2},\ldots,T_{iq}\}$ is completed, and before the set of tasks $\{T_{i+1,1},T_{i+1,2},\ldots,T_{i+1,q}\}$ is started. Therefore,

$$\omega = s + s\omega_1 \tag{5.36}$$

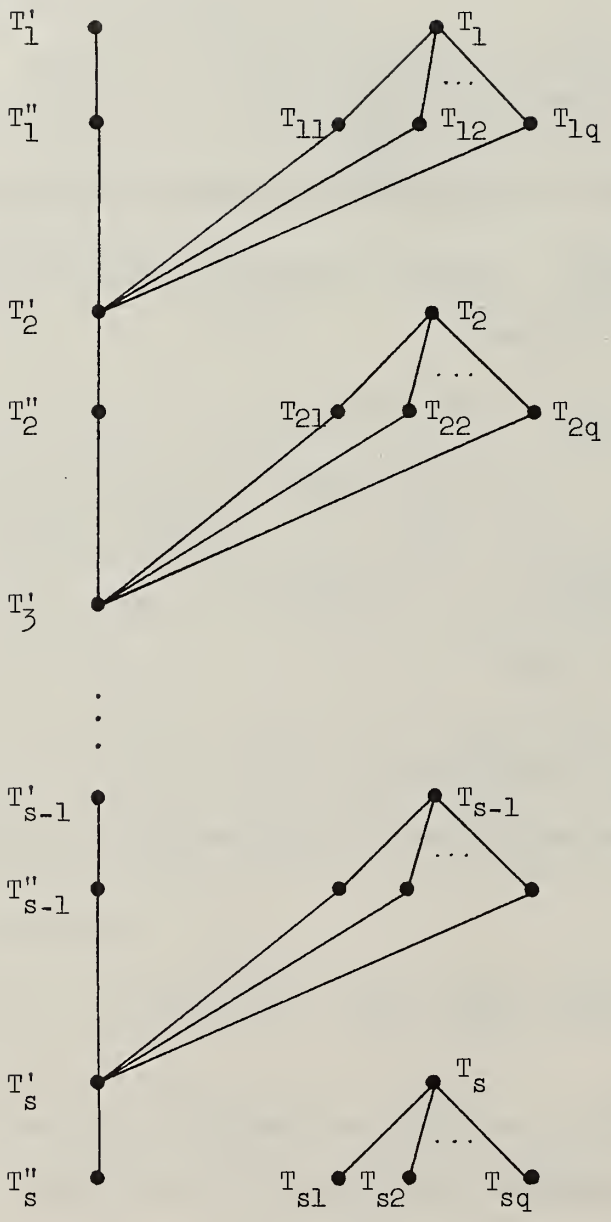


Figure 5.3 A "bad" partial order for the level algorithm.

where ω_1 is the amount of time needed to execute all the tasks in $\{T_{11}, T_{12}, \ldots, T_{1q}\}$ scheduled according to the list L_1.

Now the problem of scheduling $\{T_{11}, T_{12}, \ldots, T_{1q}\}$ is identical to the bin-packing problem [10], [15] since $R_j(T_{1k}) = 0$ for all $j \neq 1$. In fact, we can regard R_1 as the weight function, and the completion time as the number of boxes used in the bin-packing problem (cf. section 9). It is known [10], [15] that we can choose the weight function so that the number of boxes used is $\frac{17}{10} - \epsilon$ times greater than the optimal number needed. Thus, by properly choosing R_1, we have

$$\omega_1 \geq (\frac{17}{10} - \epsilon)\omega_{10} \tag{5.37}$$

where ω_{10} is the optimal time needed to finish all tasks in $\{T_{11}, T_{12}, \ldots, T_{1q}\}$ scheduled according to some list $\{T_{1i_1}, T_{1i_2}, \ldots, T_{1i_q}\}$.

Returning to the scheduling of tasks in Figure 5.3, we form the list $L_0 = (T_1, T_2, \ldots, T_s, T_1', \ldots, T_s', T_1'', \ldots, T_s'')L_1'L_2' \ldots L_s'$ where $L_k' = (T_{ki_1}, T_{ki_2}, \ldots, T_{ki_q})$. It is clear that tasks in the list $L_1', L_2', \ldots, L_s'$ will be performed in parallel. This leads to

$$\omega_0 \leq s + \omega_{10} \tag{5.38}$$

A comparison of (5.36), (5.37), and (5.38) shows

$$\frac{\omega}{\omega_0} > \frac{17}{10} s - \epsilon' \qquad \text{for } \omega_0 \gg s$$

This proves Theorem 5.7. □

5.8 Proofs of Theorems 5.8 and 5.9

(A) Proof of Theorem 5.8. We shall only prove equation (5.8). The other equation in Theorem 5.8 can be established similarly.

First we partition the set $\{1,2,\ldots,\omega\}$ into three parts:

$$A \equiv \{k \mid 1 \leq k \leq \omega, \ \exists \text{ a task } T(i) \in F_k \text{ such that } m(T(i)) > 1/2\}$$

$$B \equiv \{k \mid 1 \leq k \leq \omega, \ W(F_k) \geq 2/3\} - A$$

$$C \equiv \{1,2,\ldots,\omega\} - A - B$$

Next we define s subsets of tasks:

$$E_j \equiv \{T(i) \mid R_j(T(i)) > 1/2\} \qquad j=1,2,\ldots,s \ .$$

From the definitions of A and the E_j's, it follows that $\sum_{j=1}^{s} E_j$ has at least $|A|$ elements. Thus,

$$\sum_{j=1}^{s} |E_j| \geq |A| \ .$$

Let d, where $1 \leq d \leq s$, be such that $|E_d| = \max_j \{|E_j|\}$. Then $|E_d| \geq \frac{1}{s}|A|$. Since no two tasks in E_d can be executed simultaneously in any schedule, we must have,

$$\omega_0 \geq |E_d| \geq \frac{1}{s}|A| \tag{5.39}$$

Two more inequalities follow easily from the definitions of A, B, C.

$$\omega_0 \geq \frac{1}{s}\left(\frac{1}{2}|A| + \frac{2}{3}|B|\right) \tag{5.40}$$

$$|A| + |B| + |C| = \omega \tag{5.41}$$

We need one more inequality. For the moment, let us assume that the following is true.

Claim

$$\omega_0 \geq |C| \tag{5.42}$$

From the four equations (5.39), (5.40), (5.41), and (5.42), we can eliminate $|A|$, $|B|$, $|C|$ to obtain

$$(\frac{7}{4} s + 1)\omega_0 \geq \omega \tag{5.43}$$

which is the formula we want to prove.

The only remaining work now is to prove the validity of the claim, equation (5.42).

Lemma 5.19 Let $\ell \epsilon C$ and $T(i) \epsilon F_{\ell'}$, where $\ell' > \ell$. If for every ℓ'', $\ell < \ell'' < \ell'$, there is no $T(h) \epsilon F_{\ell''}$ satisfying $T(h) <\cdot T(i)$, then there must exist a task $T(j) \epsilon F_{\ell}$ such that $T(j) <\cdot T(i)$.

Proof Suppose otherwise. Then clearly there must be some task in F_{ℓ} that appears before T_i in the list. We shall prove that this leads to contradictions.

Case (i): There is only one task $T(k)$ in F_{ℓ} that appears before $T(i)$ in the list. It is clear that $m(T(k)) \leq 1/2$ because $\ell \epsilon C$. Furthermore, since the list is prepared by resource decreasing algorithm, we have

$$m(T(i)) \leq W(T(i)) \leq W(T(k)) \leq 1/2 \ .$$

Therefore, $T(i)$ could be done with $T(k)$. Since $T(i)$ has priority over all tasks in F_{ℓ} other than $T(k)$, $T(i)$ should be done with $T(k)$. This is a contradiction.

Case (ii): There are at least two tasks $T_k, T_{k'}$, that appear before T(i) in the list. Again we have

$$W(T(i)) \leq W(T(k)), W(T(i)) \leq W(T(k')) .$$

Thus, $W(T(i)) \leq \frac{1}{2}(W(T_k) + W(T_{k'})) \leq \frac{1}{2} \sum_{T(j)\in F_\ell} W(T(j))$

$$W(T(i)) + \sum_{T(j)\in F_\ell} W(T(j)) \leq \frac{3}{2} \sum_{T(j)\in F_\ell} W(T(j)) < \frac{3}{2} \cdot \frac{2}{3} = 1 \qquad (5.44)$$

where, in the last step, the fact that $\ell \in C$ is used.

Thus, T(i) could be done with all the T(j)'s in F_ℓ. This again contradicts the fact that $T(i) \in F_\ell$. This proves the lemma. □

Lemma 5.20 There is a chain $T(j_q) <\cdot T(j_{q-1}) <\cdot \ldots <\cdot T(j_1)$ such that, for any $\ell \in C$, there is a task $T(j_k)$ in the chain such that $T(j_k) \in F_\ell$.

Proof This chain can easily be constructed "from bottom up" by making use of Lemma 5.18. □

An immediate consequence of Lemma 5.19 is that there is a chain of length greater or equal to $|C|$. Thus, $\omega_0 \geq |C|$, which proves the chain, equation (5.42). This completes the proof of Theorem 5.8. □

(B) Proof of Theorem 5.9. We define a system of tasks as follows:

(i) $F = \{T_i, T_i', T_{ij}, T_{ij}' \mid 1 \leq i \leq s,\ 1 \leq j \leq q\}$

(ii) The partial order is defined by (see Figure 5.4):

$$T_i <\cdot T_i' <\cdot T_j <\cdot T_j' \qquad 1 \leq i < j \leq s$$

$$T_i <\cdot T_{ij},\ T_i' <\cdot T_{ij}' \qquad \forall i,j$$

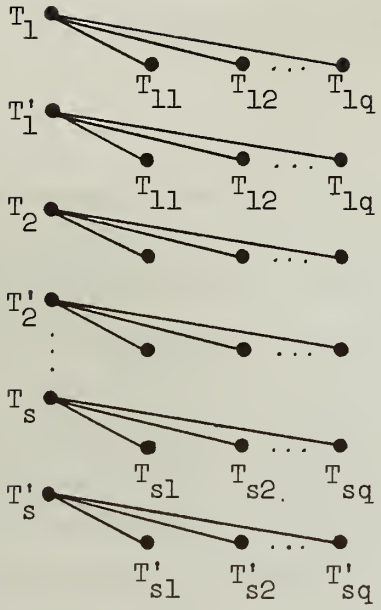


Figure 5.4 A "bad" partial order for the resource decreasing algorithm.

(iii) $\vec{R}(T_i) = (3\epsilon, 3\epsilon, \ldots, 3\epsilon)$

$\vec{R}(T'_1) = (0, \ldots, 0, \frac{1}{2} - \epsilon + \delta, 0, \ldots, 0)$ $\quad 1 \leq i \leq s$

$\vec{R}(T_{ij}) = (0, \ldots, 0, \frac{1}{2} + \epsilon, 0, \ldots, 0)$

$\vec{R}(T'_{ij}) = (0, \ldots, 0, \frac{1}{2} - \epsilon, 0, \ldots, 0)$ $\quad 1 \leq i \leq s,\ 1 \leq j \leq q$

with q an even number, and $0 < \delta < \epsilon < 1/8$.

Let L be the list generated by the resource decreasing algorithm. Then $L = L_1 L_2 \ldots L_s (T'_1 T'_2 \ldots T'_s) L'_1 L'_2 \ldots L'_s (T_1 T_2 \ldots T_s)$ where $L_i = (T_{i1}, T_{i2}, \ldots, T_{i2})$ and $L'_i = (T'_{i1}, T'_{i2}, \ldots, T'_{iq})$. For this list, $\omega = s(\frac{3q}{2} + 2)$ as can be seen from Figure 5.5.

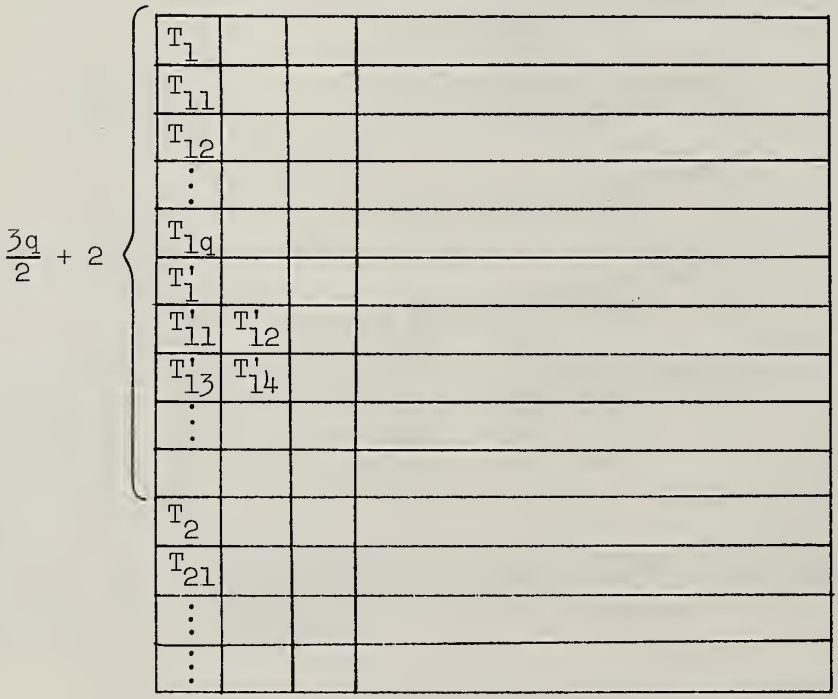


Figure 5.5 The schedule generated by the resource decreasing algorithm for the system in Figure 5.4.

On the other hand, by choosing a proper list, the tasks can all be completed in time $\omega_0 = 2s + q$ (see Figure 5.6).

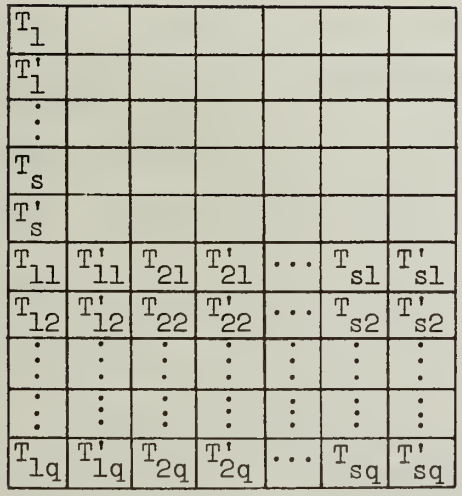


Figure 5.6 An optimal schedule for the system in Figure 5.4.

As $q \to \infty$, $\frac{\omega}{\omega_0} = \frac{s(\frac{3}{2}q+2)}{2s+q} > \frac{3}{2}s - \epsilon_0$ for any fixed $\epsilon_0 > 0$. □

5.9 The Multi-Weight Packing Problem

The proofs for the theorems in section 5 are most conveniently described in the language of a packing problem. In this section we shall define the packing problem and formulate the theorems in section 5 in this new context. The proofs of these theorems are given in Appendix B. Stronger results are now known [12].

(A) The Problem. Suppose we have an unlimited supply of boxes $B_1, B_2, \ldots$ and we want to pack objects $O_1, O_2, \ldots, O_r$ into these boxes. Associated with each object O_j is an s-dimensional weight vector $\vec{a}_j$ whose components are real numbers between 0 and 1. A box B_j can hold objects $\{O_{i_1}, O_{i_2}, \ldots, O_{i_p}\}$ if each component of the sum vector $\vec{a}_{i_1} + \vec{a}_{i_2} + \ldots + \vec{a}_{i_p}$ is no greater than 1. The problem is to pack a given set of objects into as few boxes as possible. When s = 1, this becomes the bin-packing problem which is well-studied in the literature [7].

(B) First-fit Algorithm. In the above problem, any list $L = (O_{i_1}, O_{i_2}, \ldots, O_{i_r})$, where $(i_1, i_2, \ldots, i_r)$ is a permutation of $(1, 2, \ldots, r)$, generates a packing scheme as follows:

Step (1): Take a sequence of boxes $B_1, B_2, \ldots, B_r$.

Step (2): Set $\vec{z}(B_i) = 0$ for all i.

Step (3): For $j=1,2,\ldots,r$, do the following: Find the least k such that each component of the vector $\vec{z}(B_k) + \vec{a}_{i_j}$ is less than or equal to 1. Set $\vec{z}(B_k) \leftarrow \vec{z}(B_k) + \vec{a}_{i_j}$. And we say that we have "packed" O_{i_j} into box B_k.

We shall use $N_{FF}(L)$ to denote the number of non-empty boxes after the above procedure. N_0 is defined to be the minimum of $N_{FF}(L)$ over all lists L. When s = 1, the first-fit algorithm defined above agrees with the usage of this term in the bin-packing algorithm.

(C) Connection Between Scheduling and Packing. Consider the scheduling of tasks $\{T(1),T(2),\ldots,T(r)\}$ with empty partial order $<.$, resource demand $R(T(j))$, and number of processors $n \geq r$. Let us turn it into a multi-weight packing problem via the following correspondence:

$$\begin{aligned} &T(j) : O_j \\ &\vec{R}(T(j)) : \vec{a}_j \\ &\text{completion time : number of boxes used} \end{aligned} \tag{5.45}$$

Lists: $L' = (T(i_1),\ldots,T(i_r)) : L = (O_{i_1},\ldots,O_{i_r})$.

Given any list $L' = (T(i_1),T(i_2),\ldots,T(i_r))$ we can generate a list schedule for the scheduling problem. This process can be translated via (5.45) into an algorithm for generating a packing scheme based on list $L = (O_{i_1},O_{i_2},\ldots,O_{i_r})$. If we compare this algorithm with the first-fit algorithm, we see that these two algorithms are not the same. However, it is not too difficult to see that the resulting packing schemes are the same. As a result, $\omega'_L = N_{FF}(L)$, and $\omega_0 = N_0$. This allows us to state theorems in section 5 in the following form.

Theorem 5.10 $N_{FF}(L)/N_0 \leq s + 17/20 + \epsilon$ for large N_0.

Theorem 5.11 There exists situations where $N_{FF}(L)/N_0 > s + 7/10 - \epsilon$ with arbitrary large N_0.

Theorem 5.13 For $s = 2$, $N_{FF}(L)/N_0 \leq 41/15 + \epsilon$ for large N_0.

A slightly modified version of packing problem is needed for Theorem 5.12.

5.10 Generalizations to Non-Uniform Task Lengths

When tasks lengths are allowed to be non-uniform, generalizations of our results take the following form: $n \geq r$

Let μ_0 = length of the shortest task

μ_1 = length of the longest task

Then $\omega_L/\mu_0 \leq \frac{1}{2} s((\omega_0/\mu_0)^2 + 7(\omega_0/\mu_0))$ for any list L

Furthermore, let us "pretend" that every task is of length μ_1 and construct a schedule according to level algorithm with every task taking μ_1 amount of time. This schedule can become a schedule for the actual problem since each task really needs no more than μ_1 amount of time. For this schedule $\omega/\omega_0 \leq (2s+1)\mu_1/\mu_0$. In general, this schedule can not be generated by a list.

5.11 Conclusions

We have considered several heuristic list scheduling algorithms. The bounds obtained on ω_L/ω_0 enable us to make certain conclusions. For example, we learned that a little extra work to prepare the list guarantees a much better efficiency of the multiprocessing system than an arbitrary list. We have also seen that these algorithms do not work as well as they do in the absence of "resource" constraints. It would be interesting if other simple algorithms can be found that make more efficient use of the parallel processing facilities.

Some of the results presented here have been improved, and will appear in a paper by Garey, Graham, Johnson, and Yao [12].

LIST OF REFERENCES

[1] Aho, A. V., J. E. Hopcroft, and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass., 1974.

[2] Berge, C. and A. Ghouila-Houri, Programming, Games and Transportation Networks, Wiley, 1965, p. 179.

[3] Blum, M., R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan, "Time Bounds for Selection," Journal of Computer and Systems Sciences 7:4, 1973, pp. 448-461.

[4] Chen, N. F. and C. L. Liu, "Scheduling Algorithms for Multiprocessing Computing Systems," Proceedings of the 1974 Sagamore Conference on Parallel Processing, August 1974.

[5] Chen, N. F. and C. L. Liu, "Bounds on the Critical Path Scheduling Algorithm on a Multiprocessor Computing System," to appear.

[6] Coffman, E. G. and R. L. Graham, "Optimal Scheduling for Two-Processor Systems," Acta Informatica 1:3, 1972, pp. 200-213.

[7] Drysdale, III, R. L. and F. H. Young, "Improved Divide/Sort/Merge Sorting Networks," Knox College preprint, 1973.

[8] Fujii, M., T. Kasami, and K. Ninomiya, "Optimal Sequencing of Two Equivalent Processors," SIAM Journal of Applied Mathematics 17:3, 1969, pp. 784-789.

[9] Garey, M. R. and R. L. Graham, "Bounds on Scheduling with Limited Resources," 4th Symposium on Operating System Principles, October 15-17, 1973.

[10] Garey, M. R., R. L. Graham, and J. D. Ullman, "Worst-Case Analysis of Memory Allocation Algorithms," Conference Record of ACM Symposium on Theory of Computing, 1972.

[11] Gear, C. W., private communication.

[12] Garey, M. R., R. L. Graham, D. S. Johnson, and A. C. Yao, to appear.

[13] Graham, R. L., "Bounds on Multiprocessing Anomalies and Related Packing Algorithms," AFIPS Conference Proceedings 40, 1972, pp. 205-217.

[14] Green, M. W., "Some Improvements in Non-Adaptive Sorting Algorithms," Proceedings of the 6th Annual Princeton Conference on Information Sciences and Systems, 1972, pp. 387-391.

[15] Johnson, D.S., A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, "Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms," SIAM Journal on Computing 3, 1974, pp. 299-326.

[16] Hu, T. C., "Parallel Scheduling and Assembly Line Problems," Operation Research 9:6, 1961, pp. 841-848.

[17] Knuth, D. E., The Art of Computer Programming, Volume 3 Sorting and Searching, Addison-Wesley, 1973.

[18] Knuth, D. E., The Art of Computer Programming, Volume 3 Sorting and Searching, Addison-Wesley, 1973, pp. 234-235.

[19] Liu, Jane W. S. and C. L. Liu, "Bounds on Scheduling Algorithms for Heterogeneous Computing Systems," Proceedings of the 1974 IFIP Congress, August 1974

[20] Rabin, M. O., "Solving Linear Equations by Means of Scalar Products," in Complexity of Computer Computations, edited by R. E. Miller and J. W. Thatcher, Plenum Press, 1972.

[21] Aho, A. V., J. E. Hopcroft, and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass., 1974, Section 4.6.

[22] Shafarevich, I. R., Basic Algebraic Geometry, Springer-Verlag, 1974.

[23] Tarjan, R. E., unpublished.

[24] Ullman, J. D., "Polynomial Complete Scheduling Problems," 4th Symposium on Operating System Principles, October 15-17, 1973.

[25] Van Voorhis, D. C., "Toward a Lower Bound for Sorting Networks," in Complexity of Computer Computations, edited by R. E. Miller and J. W. Thatcher, Plenum Press, 1972.

[26] Yao, F. F., private communication.

[27] Yao, A. C., "Bounds on Selection Networks," Proceedings of the 15th SWAT Conference, pp. 110-116.

[28] Yao, A. C., "On Computing the Minima of Quadratic Forms," to appear in the Proceedings of 1975 SIGACT Conference.

[29] Yao, A. C., "An $O(|E| \log \log |V|)$ Algorithm for Finding Minimum Spanning Trees," submitted to Information Processing Letters.

[30] Yao, A. C., "On Scheduling Unit-Time Tasks with Limited Resources," Proceedings of the 1974 Sagamore Conference on Parallel Processing, August 1974.

APPENDIX A
Proof of Theorem 2.4

Let $f(t,N)$ be a function (to be defined later) that satisfies $f(t,N) \geq t$. We shall construct a family of networks $E(t,N)$ called (t,N)-eliminators with the following property: Of the N output lines of $E(t,N)$ there are $f(t,N)$ designated lines among which the smallest t elements are found for any permutation of the inputs.

According to Alexeyev's upper bound (2.2), there exists a $(t,f(t,N))$-selector F (dependent on t and N) that contains $(f(t,N)-t)(1+\frac{2\hat{S}(t)}{t}) \leq 2\lceil \log\frac{(t+1)}{3} \rceil^2 f(t,N)$ comparators. We can append this network F to the (t,N)-eliminator $E(t,N)$ by making the $f(t,N)$ designated output lines of $E(t,N)$ the inputs to the network F. Figure A.1 shows such an arrangement. Clearly this gives us a (t,N)-selector.

If $g(t,N)$ is the number of comparators contained in $E(t,N)$, then the total number of comparators in the (t,N)-selector of Figure A.1 is bounded from above by

$$g(t,N) + 2\lceil \log\frac{(t+1)}{3} \rceil^2 f(t,N).$$

We have proved

Lemma A.1 $\quad U(t,N) \leq g(t,N) + 2\lceil \log\frac{(t+1)}{3} \rceil^2 f(t,N) \qquad (A1)$

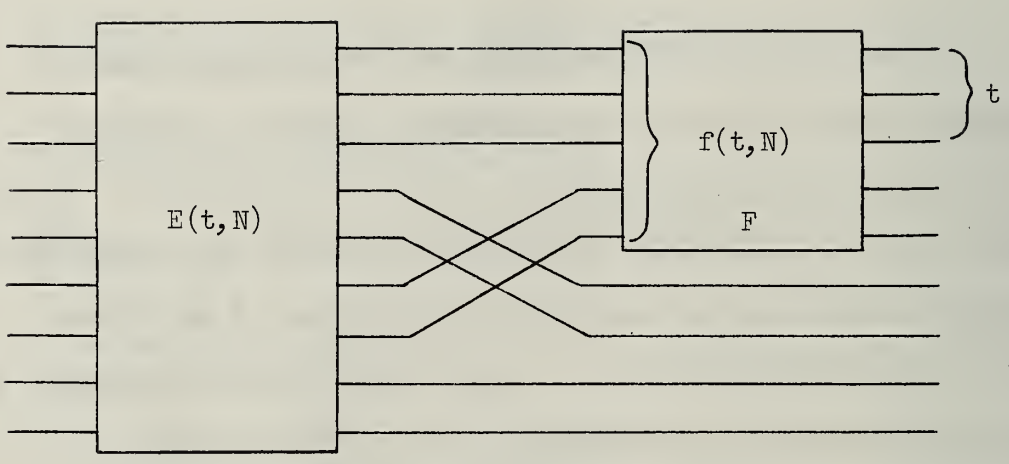
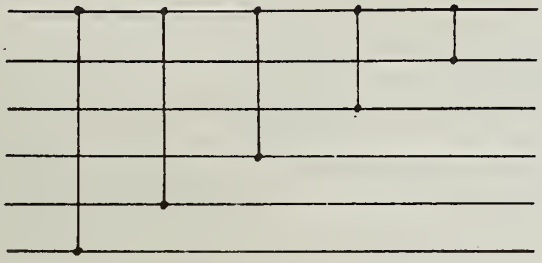


Figure A.1 Construction of a (t,N)-selector.

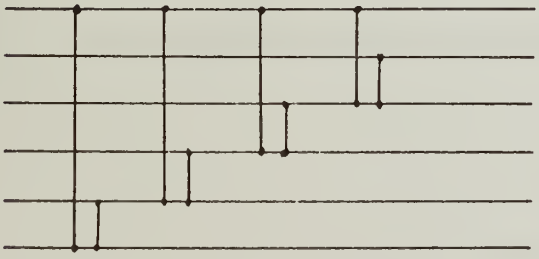
We shall now define $f(t,N)$, $E(t,N)$, and derive upper bounds for $g(t,N)$. They can then be substituted into (A1) to prove Theorem 2.4.

Network $E(t,N)$ $E(t,N)$ and $f(t,N)$ are derived inductively as follows:

(a) $E(1,N)$: $f(1,N) = 1$



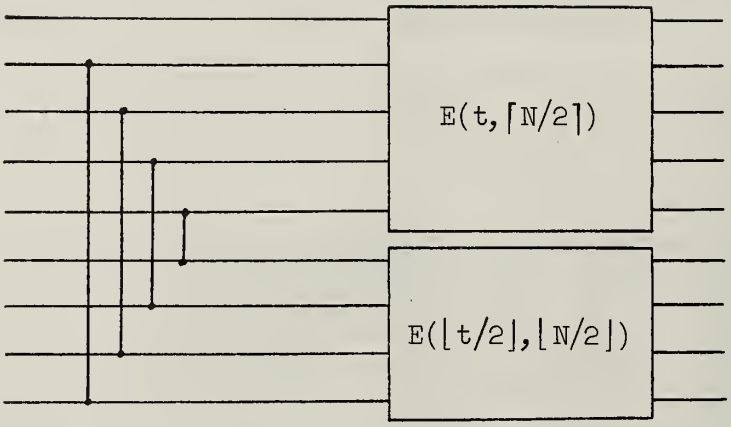
$E(2,N)$: $f(2,N) = 2$



(b) $E(t,N)$ for $t \geq 3$

(i) if $t \geq N$, $E(t,N)$ contains no comparators and $f(t,N) = N$.

(ii) if $t < N$, $E(t,N)$ is given by



and $$f(t,N) = f(t,\lceil N/2\rceil) + f(\lfloor t/2\rfloor,\lfloor N/2\rfloor) \qquad \text{(A2)}$$

From this construction, we have

$$g(t,N) \begin{cases} = 0 & \text{if } t \geq N \\ = N-1 & \text{if } t = 1 \\ = 2N-4 & \text{if } t = 2 \\ = g(t,\lceil N/2\rceil + g(\lfloor t/2\rfloor,\lfloor N/2\rfloor) + \lfloor N/2\rfloor & \text{if } 3 \leq t \leq N \end{cases} \qquad \text{(A3)}$$

From (A2) and (A3), the following lemma can be proved by induction.

Lemma A.2 $\quad g(t,N) \leq \lceil (t+1)\rceil N$

$$f(t,N) \leq 2\left[\binom{\lceil \log N\rceil}{0} + \binom{\lceil \log N\rceil}{1} + \ldots + \binom{\lceil \log N\rceil}{\lceil \log \frac{t+1}{3}\rceil}\right] \qquad \text{(A4)}$$

Substituting (A4) into (A1), we obtain

$$U(t,N) \leq \lceil \log(t+1) \rceil N + 4 \lceil \log \frac{t+1}{3} \rceil^2 \sum_{k=0}^{\lceil \log \frac{t+1}{3} \rceil} \binom{\lceil \log N \rceil}{k} \qquad (A5)$$

Now, if $t < \sqrt{N}$, $\binom{\lceil \log N \rceil}{k} \leq \binom{\lceil \log N \rceil}{\lceil \log \frac{t+1}{3} \rceil}$ for all $k=0,1,2,\ldots, \lceil \log \frac{t+1}{3} \rceil$.

Therefore (A5) implies

$$U(t,N) \leq \lceil \log(t+1) \rceil N + 4 \lceil \log \frac{t+1}{3} \rceil^3 \binom{\lceil \log N \rceil}{\lceil \log \frac{t+1}{3} \rceil}$$

for $t < \sqrt{N}$. This completes the proof of Theorem 2.4. □

APPENDIX B
Proofs for Theorems 5.11, 5.12, 5.14

Proof of Theorem 5.11 $<\cdot$ = empty, $n \geq r$, $\mu(T_i) = 1 \quad \forall i$

$$\frac{\omega_L}{\omega_0} \leq s + \frac{17}{20} + \epsilon \qquad \text{if } \omega_0 \text{ is large enough} \tag{B1}$$

As was stated in section 5.9, this problem is equivalent to a general bin-packing problem which differs from the conventional bin-packing problem only in that the weights associated with objects are now s-dimensional vectors instead of simple numbers. For a box to hold a set of objects, each component of the sum of the weight vectors must be equal to or less than 1. A first-fit algorithm can be naturally defined. It is possible to prove (see remarks in section 5.9 of the paper) that Theorem 5.11 can be formulated as:

$$\frac{N_{FF}(L)}{N_0} \leq s + \frac{17}{20} + \epsilon \qquad \text{for } N_0 \text{ large enough} \tag{B2}$$

We shall prove (B2). Consider the s-dimensional cube $\{(y_1, y_2, \ldots, y_s) \mid 0 \leq y_j \leq 1\}$. We divide it into s+2 regions:

Definition B1

$$\begin{cases} A \equiv \{(y_1, y_2, \ldots, y_s) \mid 0 \leq y_i \leq 1/2 \quad \forall i\} \\ C_j \equiv \{(y_1, y_2, \ldots, y_s) \mid 1/2 < y_i \leq 1,\ 0 \leq y_i \leq 1/2 \quad \forall i \neq j\}, \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i \leq j \leq s \\ D \equiv s \text{ dimensional cube} - A - \bigcup_{j=1}^{s} C_j \end{cases}$$

Definition B2 For $\vec{y} = (y_1, y_2, \ldots, y_s)$, let $\|\vec{y}\| = \sum_{i=1}^{s} y_i$

Definition B3 Let $\vec{y} = (y_1, \ldots, y_s)$, $\vec{y}' = (y_1', \ldots, y_s')$. $\vec{y}$ and $\vec{y}'$ are incompatible if $\max_i\{y_i + y_i'\} > 1$.

Now consider the packing scheme generated under first-fit algorithm by a list L. For each non-empty box B, let $\vec{z}(B) = \vec{a}_{j_1} + \vec{a}_{j_2} + \ldots + \vec{a}_{j_p}$ with the $\vec{a}_{j_t}$'s being the weight vectors associated with objects held in B.

Lemma B4 $\vec{z}(B_i)$ and $\vec{z}(B_j)$ are incompatible if B_i, B_j are distinct non-empty boxes.

Proof Property of first-fit algorithm. □

Lemma B5 There is at most one non-empty box B_j with $\vec{z}(B_j) \epsilon A$.

Proof If there is another B_k with $\vec{z}(B_k) \epsilon A$, then the components of $\vec{z}(B_j) + \vec{z}(B_k)$ are all no greater than 1 by Definition B1. This contradicts Lemma 4. □

The following lemma will be proved later.

Lemma B6 Let $1 \leq \ell \leq s$, and $\vec{y}_i = (y_{i1}, y_{i2}, \ldots, y_{is}) \epsilon C_i$ for $i=1,2,\ldots,\ell$. If $\vec{y}_i$ and $\vec{y}_j$ are incompatible for all $i \neq j$, then

$$\sum_{i=1}^{\ell} \|\vec{y}_i\| \geq \ell - 1/2 \qquad \text{(B3)}$$

Lemma B7 If $\vec{z}(B_{ji}) \epsilon C_i$ for $i=1,2,\ldots,\ell$, then $\sum_{i=1}^{\ell} \|z(B_{ji})\| \geq \ell - 1/2$.

Proof From Lemma 4 and Lemma B6. □

Lemma B8 If there are M distinct boxes non-empty $B_{j1}, B_{j2}, \ldots, B_{jn}$ such that $\vec{z}(B_{ji}) \epsilon C_1 \quad \forall i$, then $M/N_0 \leq \frac{17}{10} + \epsilon$ for large N_0.

Proof This is essentially a situation for the conventional bin-packing problem. The lemma is a consequence of the Garey-Graham-Ullman bound. □

We are now ready to prove (B2).

Let there be M_i boxes $\{B_j\}$ with $\vec{z}(B_j)\epsilon C_j$, and Q boxes $\{B_j\}$ with $\vec{z}(B_j)\epsilon D$. According to Lemma B5, there is at most one box B_j with $\vec{z}(B_j)\epsilon A$. Thus,

$$\sum_{i=1}^{s} M_i + Q + 1 \geq N_{FF}(L) \tag{B4}$$

Without loss of generality, assume $M_1 \geq M_2 \geq \ldots \geq M_s$. We partition the set of boxes $\mathcal{B} \equiv \{B_j \mid \vec{z}(B_j)\epsilon C_i \text{ for some } i\}$ into s disjoint categories:

category s: It has M_s groups of boxes. Each group has s boxes $B_{i_1}, B_{i_2}, \ldots, B_{i_s}$ with $B_{i_j}\epsilon C_j$ $j=1,2,\ldots,s$

category s-1: It has $M_{s-1}-M_s$ groups of boxes. Each group has s-1 boxes $B_{i_1}, B_{i_2}, \ldots, B_{i_s}$ with $B_{i_j}\epsilon C_j$ $j=1,2,\ldots,s-1$

category k: It has M_k-M_{k+1} groups of boxes. Each group has k boxes $B_{i_1}, B_{i_2}, \ldots, B_{i_k}$ with $B_{i_j}\epsilon C_j$ $j=1,2,\ldots,k$

category 1: It has M_1-M_2 groups of boxes. Each group has 1 box B_{i_j} with $B_{i_1}\epsilon C_1$ (B5)

According to Lemma B7, for a group of boxes $\{B_{i_1}, B_{i_2}, \ldots, B_{i_k}\}$ in category k, we have,

$$\sum_{j=1}^{k} \|z(B_{i_j})\| \geq k - 1/2 \tag{B6}$$

Moreover, $$\vec{z}(B) > 1 \quad \text{if} \quad \vec{z}(B)\epsilon D \tag{B7}$$

Therefore, from (B6), (B7), and (B5), we have:

$$\sum_{\text{all boxes } B_j \ldots} \|z(B_j)\| \geq \underbrace{M_s \cdot (s - 1/2) + \sum_{k=1}^{s-1} (M_k - M_{k+1})(k - 1/2)}_{\text{from boxes with } \vec{z}(B_j) \in C_i} + \underbrace{Q \cdot 1}_{\text{from boxes with } \vec{z}(B_j) \in D}$$

$$= Q + \sum_{k=2}^{s} M_k + 1/2\, M_1 \tag{B8}$$

From (B4, (B8), we have

$$\sum_{\text{all boxes } B_j} \|z(B_j)\| \geq (N_{FF}(L) - 1^{-M_1}) + \frac{1}{2} M_1 = N_{FF}(L) - 1 - \frac{1}{2} M_1$$

Hence, $N_0 \geq \frac{1}{s} \sum \|z(B_j)\| \geq \frac{1}{s}(N_{FF}(L) - 1 - \frac{1}{2} M_1)$ (B9)

$$s + \frac{1}{N_0}(1 + \frac{1}{2} M_1) \geq \frac{N_{FF}(L)}{N_0} \tag{B10}$$

Making use of Lemma B8, we have

$$s + \frac{17}{20} + \epsilon \geq \frac{N_{FF}(L)}{N_0} \quad \text{as} \quad N_0 \to \infty \,. \tag{B11}$$

This is just (B2).

Thus, Theorem 5.1 will be proved if we can show that Lemma B6 if true.

Proof of Lemma B6 We shall first show that Lemma B6 holds for the case $\ell = s$, i.e.,

$$\sum_{i=1}^{s} \|y_i\| \geq s - 1/2 \tag{B12}$$

We shall prove (B12) by induction:

(i) For s = 1, (B12) is obviously true, since $\vec{y}_i \epsilon C_1$.

(ii) Suppose (B12) is true for s = k-1, we shall prove it is also true for s = k. Since $\vec{y}_1$ and $\vec{y}_k$ are incompatible, either $y_{1k} + y_{kk} > 1$ or $y_{11} + y_{k1} > 1$. This is so because $y_{1j} + y_{kj} \leq 1$ for all $j \neq 1,k$. Without loss of generality, assume $y_{1k} + y_{kk} > 1$. (B13)

Now consider the projection of vectors $\vec{y}_1, \vec{y}_2, \ldots, \vec{y}_{k-1}$ in the space formed by the first k-1 components. Let us call these projections $\vec{y}'_1, \ldots, \vec{y}'_{k-1}$. For $i \neq j$ $(1 \leq i,j \leq k-1)$, $\vec{y}'_i$ and $\vec{y}'_j$ are obviously incompatible. By the induction hypothesis, we must have:

$$\sum_{i=1}^{k-1} \|\vec{y}'_i\| \geq (k-1) - 1/2 \qquad \text{(B14)}$$

Therefore, from (B13), (B14), we have:

$$\sum_{i=1}^{k} \|y_i\| \geq y_{1k} + y_{kk} + \sum_{i=1}^{k-1} \|y'_i\| \geq 1 + (k-1) - 1/2 \geq k - 1/2$$

This completes the induction proof for (B12).

To prove Lemma B6 for general ℓ, we need only observe that, any two vectors in the set $\{\vec{y}''_i \equiv (y_{i1}, y_{i2}, \ldots, y_{i\ell}) \mid i=1,2,\ldots,\ell\}$ are incompatible in the ℓ-dimensional cube. Thus, according to (B12, we must have $\sum_{i=1}^{\ell} \|\vec{y}''_i\| \geq \ell - 1/2$.

Therefore, $\sum_{i=1}^{\ell} \|\vec{y}_i\| \geq \sum_{i=1}^{\ell} \|\vec{y}''_i\| \geq \ell - 1/2$.

This completes the proof of Lemma B6, and Theorem 5.11. □

Proof of Theorem 5.12 To prove: $\exists$ systems such that $\frac{N_{FF}(L)}{N_0} > s + \frac{7}{10} - \epsilon$ and N_0 is large.

(A) For s = 1, this theorem is true by the Garey-Graham-Ullman theorem. Therefore, for any number α, there exist two lists of real numbers

$$L_{11} = (\lambda_1, \lambda_2, \ldots, \lambda_q)$$
$$L_{10} = (\lambda_{i_1}, \lambda_{i_2}, \ldots, \lambda_{i_q}) \quad \text{where } 0 \leq \lambda_i \leq 1, \text{ and } (i_1, i_2, \ldots, i_q) \text{ is a permutation of } (1,2,\ldots,q)$$

such that:

(i) $a_0 \geq \alpha$

(ii) $a/a_0 > \frac{17}{10} - \frac{\epsilon}{2}$ where $a_0 \equiv N_{FF}(L_{10})$, $a = N_{FF}(L_{11})$ (B15)

(B) Consider the function $p(x) \equiv [(\frac{17}{10} - \frac{\epsilon}{2})x - (s-1)]/(x+1)$ (B16)

Let α' be such that

$$p(x) > \frac{17}{10} - \epsilon \quad \forall\, x \geq \alpha' \tag{B17}$$

(C) We shall now construct an example to prove Theorem 5.2.

Consider a set of objects $\{O_{1i}, O_{jk}, O'_{jk} \mid 1 \leq i \leq q,\ 2 \leq j \leq s,\ 1 \leq k \leq a_0\}$, where a_0 and q are defined in (A). Let the s-dimensional weight vectors be defined as follows:

$$\vec{a}(O_{1i}) = (\lambda_i, \delta, \delta, \ldots\ldots\ldots\ldots, \delta, \delta) \qquad 1 \leq i \leq q$$

$$\vec{a}(O_{jk}) = (\underbrace{0,0,\ldots,0}_{j-1}, 1-\delta, 0, 0, \ldots 0)$$

$$\vec{a}(O'_{jk}) = (\underbrace{2\delta, 2\delta, \ldots, 2\delta}_{j-1}, \delta, 2\delta, 2\delta, \ldots, 2\delta) \qquad 2 \leq j \leq s,\ 1 \leq k \leq a.$$

where $\delta = \frac{1}{4sq}$.

Let $L \equiv L_2 L_2' L_3 L_3' \cdots L_s L_s' (O_{11} O_{12} \cdots O_{1q})$

$L_0 \equiv (O_{1i_1} O_{1i_2} \cdots O_{1i_q}) L_2 L_3 \cdots L_s L_2' \cdots L_s'$

where $L_j = (O_{j1} O_{j2} \cdots O_{ja_0})$

$L_j' = (O_{j1}' O_{j2}' \cdots O_{ja_0}')$ $\qquad 2 \leq j \leq s$

As before, let N_0 be the minimum of boxes used, then:

$$N_{FF}(L) = (s-1)a_0 + a$$

$$N_0 = N_{FF}(L_0) = a_0 + 1$$

It follows that

$$\frac{N_{FF}(L)}{N_0} = \frac{(s-1)a_0+a}{a_0+1} = s - 1 + \frac{a-(s-1)}{a_0+1} > s - 1 + \frac{(\frac{17}{10} - \frac{\epsilon}{2})a_0 - (s-1)}{a_0+1} \tag{B18}$$

where (B15) is used in the last step.

Let $\alpha_0 \geq \alpha \geq \alpha' \to \infty$, using (B17),

$$\frac{N_{FF}(L)}{N_0} > s - 1 + \frac{17}{10} - \epsilon = s + \frac{7}{10} - \epsilon$$

This proves Theorem 5.12. □

Proof of Theorem 5.14 When $s = 2$,

$$\frac{N_{FF}(L)}{N_0} \leq \frac{41}{15} + \epsilon \qquad \text{as} \quad N_0 \to \infty .$$

Proof We shall continue to use the notations defined earlier.

First divide the region $\{(y_1,y_2)\,|\,0 \leq y_1,y_2 \leq 1\}$ into 4 regions:

$$\begin{cases} A = \{(y_1,y_2)\,|\,0 \leq y_i \leq 1/2,\ i=1,2\} \\ C_1 = \{(y_1,y_2)\,|\,1/2 < y_1 \leq 1,\ 0 \leq y_2 \leq 1/2\} \\ C_2 = \{(y_1,y_2)\,|\,0 \leq y_1 \leq 1/2,\ 1/2 < y_2 \leq 1\} \\ D = \{(y_1,y_2)\,|\,1/2 < y_1,y_2 \leq 1\} \end{cases} \tag{B19}$$

Let us partition the set of all non-empty boxes into 6 disjoint classes:

$$\begin{cases} G_0 \equiv \{B_j\,|\,\vec{z}(B_j)\epsilon A\} \\ G_i \equiv \{B_j\,|\,\vec{z}(B_j)\epsilon C_i,\ B_j \text{ contains } \underline{\text{only one}} \text{ object}\} \quad i=1,2 \\ G_i' \equiv \{B_j\,|\,\vec{z}(B_j)\epsilon C_i,\ B_j \text{ contains } \underline{\text{at least two}} \text{ objects}\} \quad i=1,2 \\ G_3 \equiv \{B_j\,|\,\vec{z}(B_j)\epsilon D\} \end{cases} \tag{B20}$$

Definition B9 $g_0 \equiv |G_0|$, $g_i \equiv |G_i|$, $g_i' \equiv |G_i'|$ $i=1,2$, and $g_3 \equiv |G_3|$.

Lemma B10 $\|\vec{z}(B_j)\| > 1$ if $B_j \epsilon G_3$ (B21)

Proof Trivial. □

The following three lemmas follow easily from the definition of first-fit algorithm.

Lemma B11 If $B_j \epsilon G_1 \cup G_1'$, $B_k \epsilon G_2 \cup G_2'$, then

$$\|\vec{z}(B_j)\| + \|\vec{z}(B_k)\| \geq 3/2 \tag{B22}$$

Proof A special case of Lemma B7. □

Lemma B12 For each i (i=1,2), for all but one of the boxes $B_j \epsilon G_i'$,

$$[\vec{z}(B_j)]_i > 2/3 \tag{B23}$$

Proof Let i = 1. Suppose the lemma is false. Then there are two boxes $B_j, B_k \epsilon G_1$, $j < k$, such that $[\vec{z}(B_j)]_1 \leq 2/3$, $[\vec{z}(B_k)]_1 \leq 2/3$. Then B_k contains an object O_ℓ with weight $\vec{a}_\ell$ such that $(\vec{a}_\ell)_1 \leq 1/3$, $(\vec{a}_\ell)_2 \leq 1/2$. This means O_ℓ should be put into box B_j. This is a contradiction. □

Lemma B13 Let $B_{j_1}, B_{j_2}, \ldots, B_{j_\ell} \epsilon G_1'$ and $B_{k_1}, B_{k_2}, \ldots, B_{k_\ell} \epsilon G_2'$ be 2ℓ distinct non-empty boxes. Then, for at least ℓ-2 values of t,

$$\|\vec{z}(B_{j_t})\| + \|\vec{z}(B_{k_t})\| \geq 5/3 \qquad \text{(B24)}$$

Proof It is obvious that either $[\vec{z}(B_{j_t}) + \vec{z}(B_{k_t})]_1 > 1$ or $[\vec{z}(B_{j_t}) + \vec{z}(B_{k_t})]_2 > 1$. Assume that the former is true. Now, if B_{k_t} satisfies (B23), then $[\vec{z}(B_{j_t}) + \vec{z}(B_{k_t})]_2 > 2/3 \rightarrow \|\vec{z}(B_{j_t}) + \vec{z}(B_{k_t})\| \geq 1+2/3=5/3$. □

Lemma B14 This is at most one box in G_0, i.e., $g_0 \leq 1$.

Proof A special case of Lemma B5. □

Without loss of generality, assume $g_1 + g_1' \geq g_2 + g_2'$. There are three cases:

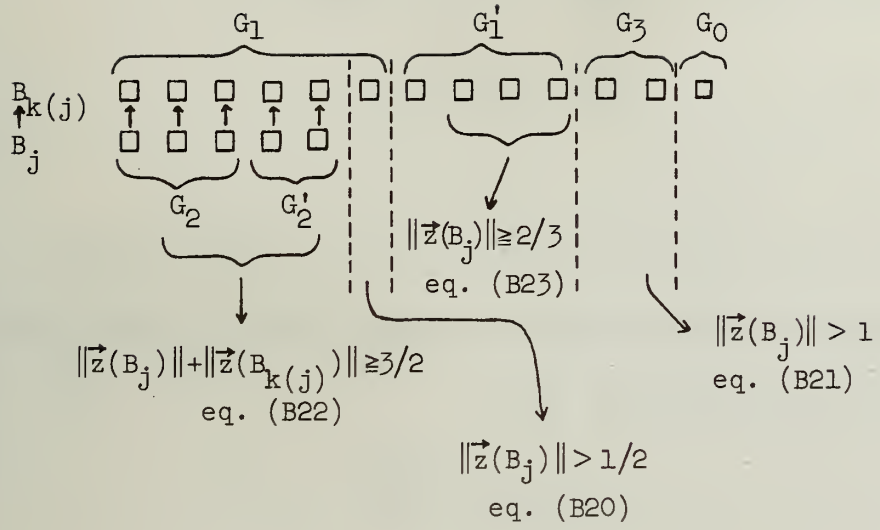
(i) $g_1 \geq g_2 + g_2'$

(ii) $g_2 + g_2' > g_1$ and $g_1 \geq g_2$

(iii) $g_2 + g_2' > g_1$ and $g_2 > g_1$

Case (i) $g_1 \geq g_2 + g_2'$

As shown in Figure (a), to each box $B_j \epsilon G_2 \cup G_2'$, associate a $B_{k(j)} \epsilon G_1$.



$$\sum_{\text{all } B_j} \|\vec{z}(B_j)\| \geq \sum_{B_j \in G_2 \cup G_2} \|\vec{z}(B_j)\| + \|\vec{z}(B_{k(j)})\| + \sum_{B_j \in G_1 - G_2 - G_2'} \|\vec{z}(B_j)\|$$

$$+ \sum_{B_j \in G_1'} \|\vec{z}(B_j)\| + \sum_{B_j \in G_3} \|\vec{z}(B_j)\|$$

$$\geq \frac{3}{2}(g_2 + g_2') + \frac{1}{2}(g_1 - g_2 - g_2') + \frac{2}{3}(g_1' - 1) + g_3$$

$$= \frac{1}{2} g_1 + \frac{2}{3} g_1' + (g_2 + g_2' + g_3) - 2/3 \qquad \text{(B25)}$$

Thus, $N_0 \geq \frac{1}{2} \sum_{\text{all } B_j} \|z(B_j)\| \geq \frac{1}{4} g_1 + \frac{1}{2}(g_2+g_2'+g_3) + \frac{1}{3} g_1' - 1/3$ (B26)

On the other hand, as in the proof for Theorem 5.1,

$$N_0 \geq (g_1+g_1') \frac{10}{17} - \epsilon \tag{B27}$$

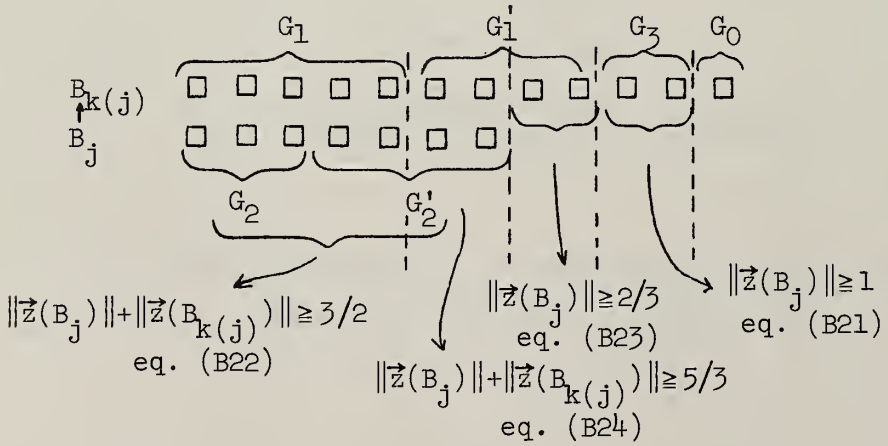
$$i + g_1 + g_2 + g_1' + g_2' + g_3 \geq N_{FF}(L) \tag{B28}$$

Furthermore, it is not difficult to prove that no two objects in $G_1 \cup G_2$ can be put in the same box, thus,

$$N_0 \geq g_1 + g_2 \tag{B29}$$

Multiplying both sides of (B26) by 2, (B27) by $\frac{17}{30}$, (B28) by 1, (B29) by $\frac{1}{6}$ and add them. We obtain, $\frac{41}{15} + \epsilon \geq \frac{N_{FF}(L)}{N_0}$ as $N_0 \to \infty$.

Case (ii) $g_2 + g_2' > g_1$ and $g_1 \geq g_2$



$$\sum_{B_j} \|\vec{z}(B_j)\| \geq \frac{3}{2} g_1 + \frac{5}{3}(g_2+g_2'-g_1-2) + \frac{2}{3}(g_1+g_1'-g_2-g_2'-1) + g_3$$

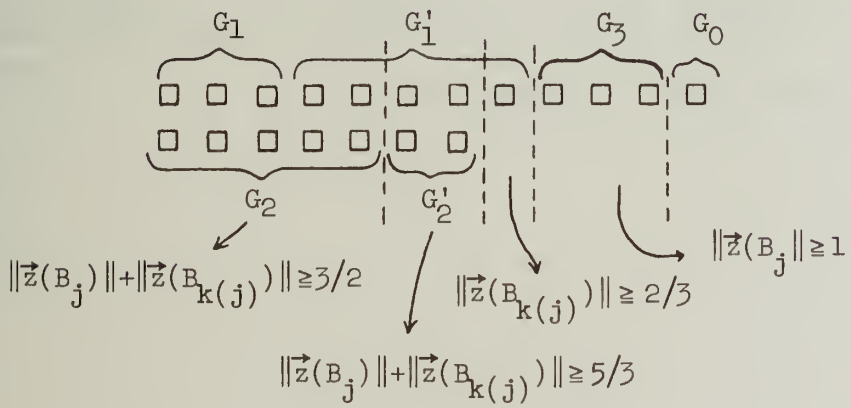
$$= \frac{1}{2} g_1 + g_2 + g_2' + \frac{2}{3} g_1' + g_3 - \frac{12}{3}$$

Therefore, $2N_0 \geq \sum_{B_j} \|z(B_j)\| \geq \frac{1}{2} g_1 + g_2 + g_2' + \frac{2}{3} g_1' + g_3 - 4$ (B30)

Note that (B30) is almost exactly the same as (B26). Since (B27), (B28), and (B29) are still valid, we can do the same as in case (i). This leads to

$$\frac{41}{15} + \epsilon \geq \frac{N_{FF}(L)}{N_0} \quad \text{as} \quad N_0 \to \infty$$

Case (iii) $g_2 + g_2' > g_1$ and $g_2 > g_1$



$$2N_0 \geq \frac{3}{2} g_2 + \frac{5}{3}(g_2'-2) + \frac{2}{3}(g_1+g_1'-g_2-g_2'-1) + g_3$$

i.e. $2N_0 \geq \frac{2}{3}(g_1+g_1') + \frac{5}{6} g_2 + g_2' + g_3 - 4$ (B31)

Now, from (B27), (B28), (B29)

$$N_0 \geq \frac{10}{17}(g_1+g_1') - \epsilon \tag{B32}$$

$$r + g_1 + g_2 + g_1' + g_2' + g_3 \geq N_{FF}(L) \tag{B33}$$

$$N_0 \geq g_1 + g_2 \tag{B34}$$

Multiply both sides of (B32) by $\frac{17}{30}$, (B33) by 1, and (B34) by $\frac{1}{6}$, then add them up. We obtain,

$$(2 + \frac{17}{30} + \frac{1}{6})N_0 \geq \frac{1}{6} g_i - (5 + \frac{17}{30} \epsilon) + N_{FF}(L)$$

$$\therefore \frac{41}{15} N_0 \geq N_{FF}(L) - (5 + \frac{17}{30} \epsilon)$$

Hence, $\frac{41}{15} + \epsilon \geq \frac{N_{FF}}{N_0}$ as $N_0 \to \infty$. This proves the theorem for case (iii). This completes the proof of Theorem 5.14. □

VITA

Andrew Chi-Chih Yao was born in Shanghai, China, on December 24, 1946. He received his B.S. in Physics from National Taiwan University in 1967, an A.M. from Harvard University in 1969, and a Ph.D. in Physics from Harvard University in 1972. Since September 1973, he has been studying Computer Science at the University of Illinois.

He served in the Chinese Air Force from July 1967 to June 1968. During his graduate study at Harvard, he visited the C.N.R.S. in Marseille, France, for one semester in 1971. From July 1972 to May 1973, he was a research associate in the Physics Department of UC Santa Barbara. He is a member of IEEE and American Mathematical Society.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-75-716	2.	3. Recipient's Accession No.
4. Title and Subtitle: A Study of Concrete Computational Complexity			5. Report Date: May 1975 6.
7. Author(s): Andrew Chi-Chih Yao			8. Performing Organization Rept. No.
9. Performing Organization Name and Address: Department of Computer Science University of Illinois at Urbana-Champaign Urbana, Illinois 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No. NSF GJ-41538
12. Sponsoring Organization Name and Address: National Science Foundation Washington, D.C.			13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstracts

In this thesis, the computational complexities of four problems are studied.

In Chapter 2, the problem of selecting t smallest numbers from a set of n by means of networks is studied. The complexity is measured both in terms of the number of comparators used, and the delay time of the network. In Chapter 3 it is shown that $O(n^2)$ functional evaluations are needed to locate the minima of quadratic forms in n variables. Chapter 4 presents an $O(e \log\log v)$ algorithm for finding a minimum spanning tree. Asymptotically, this improves over the best algorithm previously known for sparse graphs.

In the last chapter, the performances of several heuristic algorithms for a multiprocessing model (the limited-resource Garey-Graham model with unit-time-task constraints) are analyzed. In particular, it is shown that the worst-case behavior of the critical path algorithm is much better than that of an arbitrary list schedule.

17. Key Words and Document Analysis. 17a. Descriptors

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages
	20. Security Class (This Page) UNCLASSIFIED	22. Price